Adaptive compression of large vectors

Steffen Börm

April 11, 2017

Numerical algorithms for elliptic partial differential equations frequently employ error estimators and adaptive mesh refinement strategies in order to reduce the computational cost.

We can extend these techniques to general vectors by splitting the vectors into a hierarchically organized partition of subsets and using appropriate bases to represent the corresponding parts of the vectors. This leads to the concept of *hierarchical vectors*.

A hierarchical vector with m subsets and bases of rank k requires mk units of storage, and typical operations like the evaluation of norms and inner products or linear updates can be carried out in $\mathcal{O}(mk^2)$ operations.

Using an auxiliary basis, the product of a hierarchical vector and an $\mathcal{H}^2$-matrix can also be computed in $\mathcal{O}(mk^2)$ operations, and if the result admits an approximation with $\widetilde{m}$ subsets in the original basis, this approximation can be obtained in $\mathcal{O}((m+\widetilde{m})k^2)$ operations. Since it is possible to compute the corresponding approximation error exactly, sophisticated error control strategies can be used to ensure the optimal compression.

Possible applications of hierarchical vectors include the approximation of eigenvectors, optimal control problems, and time-dependent partial differential equations with moving local irregularities.

1 Introduction

We consider the standard Poisson problem

$$
\begin{aligned}
-\Delta u(x) &= f(x) && \text{for all } x \in \Omega, \\
u(x) &= g(x) && \text{for all } x \in \partial\Omega,
\end{aligned}
$$

where Ω is a Lipschitz domain with boundary $\partial\Omega$, $f : \Omega \to \mathbb{R}$ and $g : \partial\Omega \to \mathbb{R}$ are given and we are looking for the solution $u : \overline{\Omega} \to \mathbb{R}$.

If f, g and the boundary of Ω are sufficiently smooth, classical regularity theory states that the solution u will also be smooth. If only f is smooth, interior regularity results state that the solution u will at least be smooth in the interior of Ω, but may have

singularities at the boundary. If f is smooth only part of the domain, the solution u will still be smooth in the interior of this part.

Discretization schemes can take advantage of these properties to significantly reduce computational work and storage requirements: a standard finite element scheme can use fairly large elements to represent smooth parts of the solution and refine the triangulation locally close to the non-smooth parts [2, 17, 15], and sophisticated error estimation techniques [1, 9, 14, 13, 16, 18] have been developed to automatically choose parts of the mesh that should be refined. A particularly elegant approach can be developed for wavelet techniques [8] by looking for the "most important" among the (infinitely many) coefficients of the solution.

All of these techniques rely on special properties of the operators and spaces involved in the computation, e.g., coercivity of bilinear forms or local approximation estimates for finite element spaces.

Some of these requirements can be avoided by following a purely algebraic approach: instead of using a locally refined discretization, we rely on a *uniform* discretization that can represent all functions expected to appear in the algorithm sufficiently well. For the sake of simplicity, we assume that the discretization corresponds to mesh points $(x_i)_{i \in \mathcal{I}}$, where $\mathcal{I}$ is a finite index set, e.g., the set of nodal points of a finite element discretization. Each function $u \in \Omega$ then corresponds to a vector $\mathbf{u} \in \mathbb{R}^{\mathcal{I}}$ given by

$$u_i = u(x_i) \qquad \text{for all } i \in \mathcal{I}.$$

Since we are using a uniform mesh, the dimension $n = \#\mathcal{I}$ can be expected to be very large, and working with vectors $\mathbf{u} \in \mathbb{R}^{\mathcal{I}}$ directly would take too long and require too much storage.

We can significantly improve the efficiency by using *data-sparse approximations* of vectors. The compression scheme takes its cue from $\mathcal{H}^2$-matrices [12, 5, 3]: if a function $u : \Omega \to \mathbb{R}$ is smooth in a subdomain $\omega \subseteq \Omega$, e.g., due to interior regularity properties, we can approximate $u|_\omega$ by polynomials and obtain

$$u(x) \approx \sum_{\nu=1}^{k} p_\nu(x) \hat{u}_\nu \qquad \text{for all } x \in \omega,$$

where $(p_\nu)_{\nu=1}^{k}$ is a polynomial basis and $\hat{u} \in \mathbb{R}^k$ is a matching coefficient vector. For the corresponding vector $\mathbf{u}$ we have

$$u_i = u(x_i) \approx \sum_{\nu=1}^{k} p_\nu(x_i) \hat{u}_\nu \qquad \text{for all } i \in \mathcal{I} \text{ with } x_i \in \omega.$$

Introducing the subset

$$\hat{\omega} := \{i \in \mathcal{I} \ : \ x_i \in \omega\}$$

and the matrix $V \in \mathbb{R}^{\hat{\omega} \times k}$ with

$$v_{i\nu} := p_\nu(x_i) \qquad \text{for all } i \in \hat{\omega}, \ \nu \in [1 : k],$$

we can write the approximation result in the short form

$$\mathbf{u}|_{\hat{\omega}} \approx \mathbf{V}\widehat{\mathbf{u}}.$$

This approximation is only valid for indices in the subset $\hat{\omega} \subseteq \mathcal{I}$. In order to obtain an approximation for the entire vector $\mathbf{u}$, we split the index set $\mathcal{I}$ into m disjoint subsets $\hat{\omega}_1, \ldots, \hat{\omega}_m$ and approximate each subvector $\mathbf{u}|_{\hat{\omega}_j}$. The resulting approximation of $\mathbf{u}$ requires mk coefficients, and if the function u is smooth in large subdomains of Ω, we can expect $mk \ll n$.

Having a representation of $\mathbf{u}$ by mk coefficients at our disposal, we are of course interested in performing algebraic operations with these representations, e.g., computing linear combinations of compressed vectors, evaluating inner products and norms, and multiplying compressed vectors by matrices. Under suitable assumptions, all of these operations can be carried out in $\mathcal{O}(mk)$ or $\mathcal{O}(mk^2)$ operations.

Compared to standard adaptive finite element methods, this approach has several advantages:

- hierarchical vectors can be used with any matrix that can be approximated by an $\mathcal{H}^2$-matrix, e.g., matrices arising in the boundary element method or in the context of population dynamics,

- refining and coarsening a hierarchical vector only involves adding and removing subtrees of a prescribed cluster tree, no special treatment of hanging nodes or differing polynomial degrees is required,

- linear combinations and inner products of hierarchical vectors corresponding to completely different subdivisions of the index set can be computed efficiently, and

- the approximation error in all of these operations can be computed exactly.

The algorithm for the efficient multiplication of a hierarchical vector by an $\mathcal{H}^2$-matrix requires certain precomputed auxiliary matrices, and in a simple implementation the setup of these matrices would require $\mathcal{O}(nk^2)$ operations, making the method only attractive in situations where a large number of matrix-vector multiplications have to be carried out with the same $\mathcal{H}^2$-matrix, e.g., for time-dependent problems like the heat or wave equation, or for the approximation of eigenvectors by a preconditioned inverse iteration.

This disadvantage can be overcome if the differential or integral operator underlying the matrix is translation-invariant, since this property implies that matrices corresponding to translation-equivalent blocks are identical, and it can be expected that computing the auxiliary matrices only once for each equivalence class reduces the complexity to $\mathcal{O}(\log(n)k^2)$. Translation-invariance can even be exploited if it is available only in a subdomain.

2 Hierarchical vectors

In order to be able to construct counterparts of local refinement and coarsening of meshes, we introduce a hierarchy of subsets of $\mathcal{I}$.

Definition 2.1 (Labeled tree) *Let V be a finite set, let $r \in V$, let $S : V \to \mathcal{P}(V)$ be a mapping from V into the power set of V, and let $\iota : V \to M$ be a mapping from V into an arbitrary set M.*

$\mathcal{T} = (V, r, S, \iota)$ is called a (labeled) tree *if for each $v \in V$ there is exactly one sequence $v_0, v_1, \ldots, v_\ell \in V$ such that*

$$v_0 = r, \qquad V_\ell = v, \qquad v_i \in S(v_{i-1}) \qquad \text{for all } i \in [1 : \ell].$$

In this case, r is called the root *of $\mathcal{T}$ and denoted by $\mathrm{root}(\mathcal{T})$, and $S(v)$ are called the* sons *of $v \in V$ and denoted by $\mathrm{sons}(\mathcal{T}, v)$.*

For each $v \in V$, $\iota(v) \in M$ is called the label *of v and denoted by $\hat{v}$.*

Definition 2.2 (Cluster tree) *Let $\mathcal{T}_\mathcal{I} = (V, r, S, \iota)$ be a labeled tree. We call it a* cluster tree *for the index set $\mathcal{I}$ if*

- *$\hat{r} = \mathcal{I}$, i.e., if the root is labeled with $\mathcal{I}$,*

- *we have*

$$\hat{t} = \bigcup_{t' \in \mathrm{sons}(t)} \hat{t}' \qquad \text{for all } t \in V \text{ with } \mathrm{sons}(t) \neq \emptyset,$$

  i.e., the label of a cluster is contained in the union of the labels of its sons, and

- *we have*

$$t_1 \neq t_2 \Rightarrow \hat{t}_1 \cap \hat{t}_2 = \emptyset \qquad \text{for all } t \in V, \ t_1, t_2 \in \mathrm{sons}(t),$$

  i.e., different sons of the same cluster are disjoint.

If $\mathcal{T}_\mathcal{I}$ is a cluster tree, we call the elements $t \in V$ clusters *and use the short notation $t \in \mathcal{T}_\mathcal{I}$ for $t \in V$.*

Definition 2.3 (Leaves) *Let $\mathcal{T}_\mathcal{I}$ be a cluster tree. A cluster $t \in \mathcal{T}_\mathcal{I}$ is called a* leaf *of $\mathcal{T}_\mathcal{I}$ if $\mathrm{sons}(\mathcal{T}_\mathcal{I}, t) = \emptyset$.*

The set of all leaves is denoted by

$$\mathcal{L}_\mathcal{I} := \{t \in \mathcal{T}_\mathcal{I} \ : \ \mathrm{sons}(\mathcal{T}_\mathcal{I}, t) = \emptyset\}.$$

Remark 2.4 (Leaf partition) *A simple induction yields that the set*

$$\{\hat{t} \ : \ t \in \mathcal{L}_\mathcal{I}\}$$

is a disjoint partition of $\mathcal{I}$, so we can describe a vector $x \in \mathbb{R}^\mathcal{I}$ uniquely by defining its restrictions

$$x|_{\hat{t}} \in \mathbb{R}^{\hat{t}} \qquad \text{for all } t \in \mathcal{L}_\mathcal{I}.$$

Cluster trees for arbitrary index sets $\mathcal{I}$ can be constructed by fairly general algorithms usually based on recursively splitting a given subset into a number of disjoint subsets. If the indices correspond to geometric objects, e.g., points in a finite element mesh, these algorithms can ensure that clusters contain indices that are "geometrically close" to each other [10, 4].

In practical applications, it may be necessary to use different cluster trees to represent different vectors, e.g., to implement adaptive refinement towards moving singularities. In order to keep the corresponding algorithms simple and still be able to handle varying cluster trees, we use a reference tree $\mathcal{T}_\mathcal{I}$ that remains fixed and choose *subtrees* $\mathcal{T}_x$ to represent vectors $x \in \mathbb{R}^\mathcal{I}$.

Definition 2.5 (Subtree) *Let $\mathcal{T}_\mathcal{I}$ be a cluster tree for $\mathcal{I}$. A second cluster tree $\mathcal{T}_x$ for $\mathcal{I}$ is called a* subtree *of $\mathcal{T}_\mathcal{I}$ if*

- root$(\mathcal{T}_x) = $ root$(\mathcal{T}_\mathcal{I})$,

- *we have*

$$t \in \mathcal{T}_\mathcal{I} \qquad\qquad\qquad \text{for all } t \in \mathcal{T}_x, \text{ and}$$

- *we have*

$$\text{sons}(\mathcal{T}_x, t) \neq \emptyset \Rightarrow \text{sons}(\mathcal{T}_x, t) = \text{sons}(\mathcal{T}_\mathcal{I}, t) \qquad \text{for all } t \in \mathcal{T}_x,$$

  i.e., non-leaf clusters have the same sons in $\mathcal{T}_x$ and $\mathcal{T}_\mathcal{I}$.

If $\mathcal{T}_x$ is a subtree of $\mathcal{T}_\mathcal{I}$, we denote its leaves by $\mathcal{L}_x$.

The smallest subtree $\mathcal{T}_x$ of $\mathcal{T}_\mathcal{I}$ consists only of the root $r = \text{root}(\mathcal{T}_\mathcal{I})$, with the root a leaf of $\mathcal{T}_x$. The largest subtree is $\mathcal{T}_\mathcal{I}$ itself.

Due to Remark 2.4, the leaves of a subtree $\mathcal{T}_x$ also define a disjoint partition of $\mathcal{I}$, but this partition can be significantly coarser than the one corresponding to the leaves of $\mathcal{T}_\mathcal{I}$.

Given a partition of $\mathcal{I}$, we now turn our attention to systems of bases that can be used to represent the subvectors $x|_{\hat{t}}$ corresponding to the leaves $t \in \mathcal{T}_x$ of a cluster tree.

In order to be able to "refine" a given hierarchical vector, i.e., to subdivide leaves of the corresponding subtree, we require these bases to be *nested*, i.e., if $x|_{\hat{t}}$ can be represented in the basis corresponding to the cluster $t \in \mathcal{T}_\mathcal{I}$, it has to be possible to represent $x|_{\hat{t}'}$ in the basis corresponding to its sons $t' \in \text{sons}(\mathcal{T}_\mathcal{I}, t)$.

Definition 2.6 (Cluster basis) *Let $k \in \mathbb{N}$ and let $(V_t)_{t \in \mathcal{T}_\mathcal{I}}$ be a family of matrices such that $V_t \in \mathbb{R}^{\hat{t} \times k}$ for all $t \in \mathcal{T}_\mathcal{I}$.*

If there is a family $(E_t)_{t \in \mathcal{T}_\mathcal{I}}$ of matrices satisfying

$$V_t|_{\hat{t}' \times k} = V_{t'} E_{t'} \qquad\qquad \text{for all } t \in \mathcal{T}_\mathcal{I}, \ t' \in \text{sons}(\mathcal{T}_\mathcal{I}, t), \qquad\qquad (2.1)$$

we call (V, E) a cluster basis *of rank k for the cluster tree $\mathcal{T}_\mathcal{I}$. The matrices $(E_t)_{t \in \mathcal{T}_\mathcal{I}}$ are called* transfer matrices.

We simply write $(V_t)_{t \in \mathcal{T}_\mathcal{I}}$ as an abbreviation and introduce the transfer matrices if they are required.

procedure refine(t, **var** x);
if sons($\mathcal{T}_\mathcal{I}, t) \neq \emptyset$ **then begin**
   Add sons($\mathcal{T}_\mathcal{I}, t$) as new leaves to $\mathcal{T}_x$;
   for $t' \in$ sons($\mathcal{T}_\mathcal{I}, t$) **do**
     $\hat{x}_{t'} \leftarrow E_{t'}\hat{x}_t$
end

Figure 1: Refining a hierarchical vector

Definition 2.7 (Hierarchical vector) *Let $\mathcal{T}_x$ be a subtree of $\mathcal{T}_\mathcal{I}$, let (V, E) be a cluster basis for $\mathcal{T}_\mathcal{I}$.*

A vector $x \in \mathbb{R}^\mathcal{I}$ is called a hierarchical vector *corresponding to $\mathcal{T}_x$ and (V, E) if there is a family $(\hat{x}_t)_{t \in \mathcal{T}_x}$ such that*

$$x|_{\hat{t}} = V_t \hat{x}_t \qquad\qquad for\ all\ t \in \mathcal{L}_x. \qquad (2.2)$$

In this case, we call $(\hat{x}_t)_{t \in \mathcal{T}_x}$ the hierarchical coefficients *for x.*

In our setting, the leaves of the subtree $\mathcal{T}_x$ play the role of the mesh used to represent a function. Locally refining the mesh corresponds to choosing a leaf $t \in \mathcal{L}_x$ with sons($\mathcal{T}_\mathcal{I}, t) \neq \emptyset$ and adding sons($\mathcal{T}_\mathcal{I}, t$) to the subtree. Due to (2.2) and (2.1), we have

$$x|_{\hat{t}'} = (V_t \hat{x}_t)|_{\hat{t}'} = V_t|_{\hat{t}' \times k} \hat{x}_t = V_{t'} E_{t'} \hat{x}_t \qquad\qquad for\ all\ t' \in \text{sons}(\mathcal{T}_\mathcal{I}, t),$$

so the equation

$$\hat{x}_{t'} := E_{t'} \hat{x}_t \qquad\qquad \text{for all } t' \in \text{sons}(\mathcal{T}_\mathcal{I}, t)$$

provides us with hierarchical coefficients for the refined tree. The procedure is summarized in Figure 1.

We can use this procedure to add a hierarchical vector x with subtree $\mathcal{T}_x$ to another hierarchical vector y with a different subtree $\mathcal{T}_y$, as long as $\mathcal{T}_x$ and $\mathcal{T}_y$ are subtrees of $\mathcal{T}_\mathcal{I}$: assume that a cluster $t \in \mathcal{T}_x \cap \mathcal{T}_y$ is given.

  1. If $t \in \mathcal{L}_x$ and $t \in \mathcal{L}_y$ holds, we can simply add $\hat{x}_t$ and $\hat{y}_t$.

  2. If $t \notin \mathcal{L}_x$ and $t \notin \mathcal{L}_y$, we consider sons($\mathcal{T}_\mathcal{I}, t$) = sons($\mathcal{T}_x, t$) = sons($\mathcal{T}_y, t$) recursively.

  3. If $t \in \mathcal{L}_x$ and $t \notin \mathcal{L}_y$, we use (2.1) to obtain temporary coefficient vectors $\hat{x}_{t'} = E_{t'} \hat{x}_t$ that can be added recursively to y.

  4. If $t \notin \mathcal{L}_x$ and $t \in \mathcal{L}_y$, we apply refine to y and proceed as in case 2.

Based on this approach, the update $y \leftarrow y + \alpha x$ can be performed by the recursive algorithm given in Figure 2. The procedure add_leaf is used to handle the cases 1 and 3, while the procedure add takes care of the cases 2 and 4.

After the procedure add has been completed, $\mathcal{T}_x$ is a subtree of $\mathcal{T}_y$ and the sum of αx and y is represented exactly by the, possibly refined, hierarchical vector y.

6

```
procedure add_leaf(t, ẑ_t, var y);              procedure add(t, α, x, var y);
if sons(𝒯_y, t) = ∅ then                        if sons(𝒯_x, t) = ∅ then
   ŷ_t ← ŷ_t + ẑ_t                                 add_leaf(t, αx̂_t, y);
else                                            else begin
   for t' ∈ sons(𝒯_y, t) do begin                 if sons(𝒯_y, t) = ∅ then
      ẑ_t' ← E_t' ẑ_t;                                refine(t, y);
      add_leaf(t', ẑ_t', y)                        for t' ∈ sons(𝒯_x, t) do
   end;                                               add(t', α, x, y)
                                                end
```

Figure 2: Adding a hierarchical vector x to a hierarchical vector y, refining the tree $\mathcal{T}_y$ as required

Remark 2.8 (Complexity) *Since we only switch to the sons $t' \in \text{sons}(\mathcal{T}_\mathcal{I}, t)$ of a cluster t if either $\text{sons}(\mathcal{T}_x, t)$ or $\text{sons}(\mathcal{T}_y, t)$ are not empty, the procedure* **add** *given in Figure 2 requires $\mathcal{O}(k^2(\#\mathcal{T}_x + \#\mathcal{T}_y))$ operations.*

We can follow a similar approach to compute inner products and norms of hierarchical vectors: let $x, y \in \mathbb{R}^\mathcal{I}$ be hierarchical vectors with subtrees $\mathcal{T}_x$ and $\mathcal{T}_y$. In order to compute the inner product

$$\langle x, y \rangle = \sum_{i \in \mathcal{I}} x_i y_i,$$

we can split $\mathcal{I}$ into subsets and consider sub-products

$$\langle x, y \rangle_t := \sum_{i \in \hat{t}} x_i y_i$$

corresponding to clusters $t \in \mathcal{T}_\mathcal{I}$. For $t = \text{root}(\mathcal{T}_\mathcal{I}) = \text{root}(\mathcal{T}_x) = \text{root}(\mathcal{T}_y)$, we obtain the full inner product $\langle x, y \rangle$.

Let $t \in \mathcal{T}_\mathcal{I}$. If $t \in \mathcal{L}_x$ and $t \in \mathcal{L}_y$, we have

$$x|_{\hat{t}} = V_t \hat{x}_t, \qquad\qquad y|_{\hat{t}} = V_t \hat{y}_t$$

and find

$$\langle x, y \rangle_t = \sum_{i \in \hat{t}} x_i y_i = \sum_{i \in \hat{t}} (V_t \hat{x}_t)_i (V_t \hat{y}_t)_i = \hat{x}_t^* V_t^* V_t \hat{y}_t. \tag{2.3}$$

The products $C_t := V_t^* V_t$ required to evaluate this expression are small $k \times k$ matrices that can be prepared using the recursion

$$C_t = \begin{cases} V_t^* V_t & \text{if } \text{sons}(\mathcal{T}_\mathcal{I}, t) = \emptyset, \\ \sum_{t' \in \text{sons}(t)} E_{t'}^* C_{t'} E_{t'} & \text{otherwise} \end{cases} \qquad \text{for all } t \in \mathcal{T}_\mathcal{I} \tag{2.4}$$

due to (2.1). With these matrices, the inner product (2.3) can be evaluated in $\mathcal{O}(k^2)$ operations.

```
function dot_leaf(t, ẑ_t, y) : real;            function dot(t, x, y) : real;
if sons(𝒯_y, t) = ∅ then                        if sons(𝒯_x, t) = ∅ then
    return ẑ*_t C_t ŷ_t                             return dot_leaf(t, x̂_t, y)
else begin                                       else if sons(𝒯_y, t) = ∅ then
  γ ← 0;                                             return dot_leaf(t, ŷ_t, x)
  for t′ ∈ sons(𝒯_y, t) do begin                 else begin
    ẑ_t′ ← E_t′ ẑ_t;                               γ ← 0;
    γ ← γ + dot_leaf(t′, ẑ_t′, y);                 for t′ ∈ sons(𝒯_x, t) do
  end;                                               γ ← γ + dot(t′, x, y);
  return γ                                         return γ
end                                              end
```

Figure 3: Compute the inner product of two hierarchical vectors x and y

If $t \notin \mathcal{L}_x$ and $t \notin \mathcal{L}_y$, we can use Definition 2.2 to get

$$\langle x, y \rangle_t = \sum_{t' \in \mathrm{sons}(\mathcal{T}_\mathcal{I}, t)} \langle x, y \rangle_{t'},$$

i.e., we can compute the products for the sons $t' \in \mathrm{sons}(\mathcal{T}_\mathcal{I}, t)$ by recursion and add the results.

If $t \in \mathcal{L}_x$ and $t \in \mathcal{T}_y \setminus \mathcal{L}_y$, we can again use Definition 2.2 and (2.1) to obtain

$$\langle x, y \rangle_t = \sum_{t' \in \mathrm{sons}(\mathcal{T}_y, t)} \langle x, y \rangle_{t'} = \sum_{t' \in \mathrm{sons}(\mathcal{T}_y, t)} \langle V_t \hat{x}_t, y \rangle_{t'}$$

$$= \sum_{t' \in \mathrm{sons}(t)} \langle V_{t'} E_{t'} \hat{x}_t, y \rangle_{t'} = \sum_{t' \in \mathrm{sons}(t)} \langle V_{t'} \hat{z}_{t'}, y \rangle_{t'}$$

with the auxiliary vectors $\hat{z}_{t'} = E_{t'} \hat{x}_t$. We can repeat this procedure recursively until we reach a leaf $t \in \mathcal{L}_y$ and then use C_t as before.

If $t \in \mathcal{T}_x \setminus \mathcal{L}_x$ and $t \in \mathcal{L}_y$, we can use the same approach with auxiliary vectors $\hat{z}_{t'} = E_{t'} \hat{y}_t$.

The resulting recursive algorithm is summarized in Figure 3. Due to $\|x\| = \sqrt{\langle x, x \rangle}$, we can also use this function to compute the norm of a hierarchical vector.

Remark 2.9 (Complexity) *Preparing C_t for all $t \in \mathcal{T}_\mathcal{I}$ requires $\mathcal{O}(k^2 \#\mathcal{I})$ operations for the leaves and $\mathcal{O}(k^3 \#\mathcal{T}_\mathcal{I})$ operations for the non-leaf clusters [3, Section 5.3].*

Once these matrices have been prepared, the procedure **dot** *given in Figure 3 requires $\mathcal{O}(k^2(\#\mathcal{T}_x + \#\mathcal{T}_y))$ operations.*

3 Coarsening

Adding two hierarchical vectors x and y using the procedure add given in Figure 2 will yield a new vector with a refined cluster tree that contains both $\mathcal{T}_x$ and $\mathcal{T}_y$ as subtrees.

This tree may not be optimal, as can be seen by considering the extreme example of adding x and $-x$ and obtaining the zero vector that can obviously be expressed by the minimal subtree of $\mathcal{T}_\mathcal{I}$.

In order to keep the computational complexity as low as possible, we introduce an algorithm that does the opposite of refining a subtree, i.e., coarsening the tree. Where the procedure `refine` given in Figure 1 splits a leaf into sons, the new `coarsen` procedure merges sons into a new leaf.

If we would use this procedure only in situations where it does not change the vector at all, it would be of very limited use. It makes more sense to consider situations where coarsening yields a reasonably good *approximation* of the original vector.

In order to devise a reliable algorithm, we have to investigate the approximation errors introduced by hierarchical vectors. We are interested in nothing less but the *best* approximation of a given vector, and this best approximation with respect to the Euclidean norm is given by an orthogonal projection. These projections are immediately available to us if we have an orthonormal basis at our disposal.

Definition 3.1 (Isometric cluster basis) *A cluster basis $(Q_t)_{t\in\mathcal{T}_\mathcal{I}}$ is called* isometric *if we have*

$$Q_t^* Q_t = I \qquad\qquad \text{for all } t \in \mathcal{T}_\mathcal{I}. \qquad (3.1)$$

There is an efficient algorithm that can turn any cluster basis into an isometric cluster basis without any change in its approximation properties [3, Section 5.4], so requiring a cluster basis to be isometric is not a significant restriction.

For an isometric basis, a simple computation yields

$$\|x - Q_t y\|^2 = \|x - Q_t Q_t^* x\|^2 + \|y - Q_t^* x\|^2 \qquad \text{for all } t \in \mathcal{T}_\mathcal{I},\ x \in \mathbb{R}^{\hat{t}},\ y \in \mathbb{R}^k, \qquad (3.2)$$

and we conclude that $Q_t Q_t^* x$ is the best approximation of x in the range of Q_t.

Just as `refine` splits a given leaf cluster t into its sons $t' \in \mathrm{sons}(\mathcal{T}_\mathcal{I}, t)$, we are looking for an algorithm that merges the sons $t' \in \mathrm{sons}(\mathcal{T}_\mathcal{I}, t)$ into the father t. We assume that $t \in \mathcal{T}_x$ is given in such a way that all of its sons $t' \in \mathrm{sons}(\mathcal{T}_x, t)$ are leaves of $\mathcal{T}_x$. To keep the presentation simple, we consider only the case of a binary tree, i.e., $\#\mathrm{sons}(\mathcal{T}_x, t) = 2$ and $\mathrm{sons}(\mathcal{T}_x, t) = \{t_1, t_2\}$. Since the sons are assumed to be leaves of $\mathcal{T}_x$, Definition 2.7 yields

$$x|_{\hat{t}} = \begin{pmatrix} x|_{\hat{t}_1} \\ x|_{\hat{t}_2} \end{pmatrix} = \begin{pmatrix} Q_{t_1} \hat{x}_{t_1} \\ Q_{t_2} \hat{x}_{t_2} \end{pmatrix}.$$

We want t to become a leaf, so we have to find a coefficient vector $\hat{x}_t$ with $x|_{\hat{t}} \approx Q_t \hat{x}_t$. Due to (3.2), the best choice is given by the orthogonal projection, i.e.,

$$\hat{x}_t := Q_t^* x|_{\hat{t}}. \qquad (3.3)$$

Computing $\hat{x}_t$ by this equation would be very inefficient, since it would require us to first construct the entire vector $x|_{\hat{t}}$ and then approximate it again. In order to reduce the

```
procedure coarsen(t, var x);
if t' ∈ L_x for all t' ∈ sons(T_x, t) then begin
   x̂_t ← 0;
   for t' ∈ sons(T_I, t) do begin
      x̂_t ← x̂_t + F*_{t'} x̂_{t'};
      Remove t' from T_x
   end
end
```

<div align="center">Figure 4: Coarsening a hierarchical vector</div>

number of operations, we rely on the nested structure (2.1) of the cluster basis: denoting the transfer matrices for $(Q_t)_{t \in T_I}$ by $(F_t)_{t \in T_I}$, we have

$$Q_t = \begin{pmatrix} Q_{t_1} F_{t_1} \\ Q_{t_2} F_{t_2} \end{pmatrix}$$

and find

$$\hat{x}_t = Q_t^* x|_{\hat{t}} = \begin{pmatrix} F_{t_1}^* Q_{t_1}^* & F_{t_2}^* Q_{t_2}^* \end{pmatrix} \begin{pmatrix} Q_{t_1} \hat{x}_{t_1} \\ Q_{t_2} \hat{x}_{t_2} \end{pmatrix}$$

$$= \begin{pmatrix} F_{t_1}^* & F_{t_2}^* \end{pmatrix} \begin{pmatrix} Q_{t_1}^* Q_{t_1} \hat{x}_{t_1} \\ Q_{t_2}^* Q_{t_2} \hat{x}_{t_2} \end{pmatrix} = \begin{pmatrix} F_{t_1}^* & F_{t_2}^* \end{pmatrix} \begin{pmatrix} \hat{x}_{t_1} \\ \hat{x}_{t_2} \end{pmatrix} = F_{t_1}^* \hat{x}_{t_1} + F_{t_2}^* \hat{x}_{t_2}.$$

Using this equation, we can compute the optimal $\hat{x}_t$ given by (3.3) efficiently. The coarsening procedure is summarized in Figure 4.

Remark 3.2 (Complexity) *If there is a constant C_{sn} such that $\# \operatorname{sons}(T_I, t) \leq C_{sn}$ holds for all $t \in T_I$, the procedures* **refine** *and* **coarsen** *require only $\mathcal{O}(k^2)$ operations.*

In order to obtain an adaptive algorithm, we have to be able to control the error introduced by coarsening steps. We define

$$\widehat{Q}_t := \begin{pmatrix} F_{t_1} \\ F_{t_2} \end{pmatrix} \in \mathbb{R}^{(2k) \times k}$$

and find that (2.1) takes the form

$$Q_t = \begin{pmatrix} Q_{t_1} & \\ & Q_{t_2} \end{pmatrix} \widehat{Q}_t.$$

The error can be written as

$$x|_{\hat{t}} - Q_t Q_t^* x|_{\hat{t}} = \begin{pmatrix} Q_{t_1} \hat{x}_{t_1} \\ Q_{t_2} \hat{x}_{t_2} \end{pmatrix} - \begin{pmatrix} Q_{t_1} & \\ & Q_{t_2} \end{pmatrix} \widehat{Q}_t \widehat{Q}_t^* \begin{pmatrix} Q_{t_1}^* & \\ & Q_{t_2}^* \end{pmatrix} \begin{pmatrix} Q_{t_1} \hat{x}_{t_1} \\ Q_{t_2} \hat{x}_{t_2} \end{pmatrix}$$

$$= \begin{pmatrix} Q_{t_1} & \\ & Q_{t_2} \end{pmatrix} \left[\begin{pmatrix} \hat{x}_{t_1} \\ \hat{x}_{t_2} \end{pmatrix} - \widehat{Q}_t \widehat{Q}_t^* \begin{pmatrix} \hat{x}_{t_1} \\ \hat{x}_{t_2} \end{pmatrix} \right].$$

Since Q_{t_1} and Q_{t_2} are isometric matrices, they leave the Euclidean norm unchanged and we conclude

$$\|x|_{\hat{t}} - Q_t Q_t^* x|_{\hat{t}}\| = \left\| \begin{pmatrix} \hat{x}_{t_1} \\ \hat{x}_{t_2} \end{pmatrix} - \widehat{Q}_t \widehat{Q}_t^* \begin{pmatrix} \hat{x}_{t_1} \\ \hat{x}_{t_2} \end{pmatrix} \right\|.$$

In theory, this equation allows us to evaluate the error explicitly in $\mathcal{O}(k^2)$ operations. In practice, however, we are subtracting two vectors with, hopefully, very similar entries, so we have to expect rounding errors to influence the result significantly.

We can avoid this problem by introducing a suitable auxiliary matrix: since Q_t, Q_{t_1} and Q_{t_2} are isometric, we have

$$I = Q_t^* Q_t = \widehat{Q}_t^* \begin{pmatrix} Q_{t_1}^* & \\ & Q_{t_2}^* \end{pmatrix} \begin{pmatrix} Q_{t_1} & \\ & Q_{t_2} \end{pmatrix} \widehat{Q}_t = \widehat{Q}_t^* \widehat{Q}_t,$$

so the matrix $\widehat{Q}_t$ is also isometric. This means that we can extend it to an orthonormal basis, i.e., we can find an isometric matrix $\widehat{P}_t \in \mathbb{R}^{(2k) \times k}$ such that

$$\begin{pmatrix} \widehat{Q}_t & \widehat{P}_t \end{pmatrix} \in \mathbb{R}^{(2k) \times (2k)} \tag{3.4}$$

is orthogonal and square, e.g., by computing the Householder factorization of $\widehat{Q}_t$ and accumulating the elementary reflections. This implies

$$I = \begin{pmatrix} \widehat{Q}_t & \widehat{P}_t \end{pmatrix} \begin{pmatrix} \widehat{Q}_t^* \\ \widehat{P}_t^* \end{pmatrix} = \widehat{Q}_t \widehat{Q}_t^* + \widehat{P}_t \widehat{P}_t^*, \qquad I - \widehat{Q}_t \widehat{Q}_t^* = \widehat{P}_t \widehat{P}_t^*,$$

and we conclude

$$\begin{pmatrix} \hat{x}_{t_1} \\ \hat{x}_{t_2} \end{pmatrix} - \widehat{Q}_t \widehat{Q}_t^* \begin{pmatrix} \hat{x}_{t_1} \\ \hat{x}_{t_2} \end{pmatrix} = \widehat{P}_t \widehat{P}_t^* \begin{pmatrix} \hat{x}_{t_1} \\ \hat{x}_{t_2} \end{pmatrix}.$$

Since $\widehat{P}_t$ is isometric, the first factor on the right-hand side does not influence the norm and we have proven the following result:

Theorem 3.3 (Coarsening error) *The matrices* $(\widehat{P}_t)_{t \in \mathcal{T}_\mathcal{I} \setminus \mathcal{L}_\mathcal{I}}$ *defined by (3.4) satisfy*

$$\|x|_{\hat{t}} - Q_t Q_t^* x|_{\hat{t}}\| = \left\| \widehat{P}_t^* \begin{pmatrix} \hat{x}_{t_1} \\ \hat{x}_{t_2} \end{pmatrix} \right\| \qquad \text{for all } t \in \mathcal{T}_\mathcal{I} \setminus \mathcal{L}_\mathcal{I}, \ x|_{\hat{t}} = \begin{pmatrix} Q_{t_1} \hat{x}_{t_1} \\ Q_{t_2} \hat{x}_{t_2} \end{pmatrix}. \tag{3.5}$$

Remark 3.4 (Implementation and complexity) *To obtain a fast and robust algorithm, we can construct* $\widehat{P}_t$ *in (3.4) by applying k Householder reflections $P_1, \ldots, P_k$ to triangularize $\widehat{Q}_t$, i.e., to obtain $P_k P_{k-1} \ldots P_1 \widehat{Q}_t = R$ with an upper triangular matrix $R \in \mathbb{R}^{2k \times k}$. Now $\widehat{P}_t$ consists of the last k columns of $P_1^* P_2^* \ldots P_k^*$. If we compute*

$$P_k P_{k-1} \ldots P_1 \begin{pmatrix} \hat{x}_{t_1} \\ \hat{x}_{t_2} \end{pmatrix}, \tag{3.6}$$

we find the error vector in the last k rows of the result.

By choosing the signs in the construction of the Householder vectors correctly, it is possible to ensure that the first k columns of $P_1^* P_2^* \ldots P_k^*$ coincide with $\widehat{Q}_t$. In this case, we can find the coefficient vector $\hat{x}_t = Q_t^* x|_{\hat{t}}$ in the first k components of (3.6) without any additional work.

If we assume that constructing a Householder vector of dimension $n \in \mathbb{N}$ and applying the corresponding reflection takes not more than $C_{qr} n$ operations, finding the k Householder reflections for triangularizing the $(2k) \times k$-matrix $\widehat{Q}_t$ takes not more than $C_{qr}(2k)k^2 = 2C_{qr}k^3$ operations and applying $\widehat{P}_t^*$ to a vector takes not more than $C_{qr}(2k)k = 2C_{qr}k^2$.

In the general case, i.e., if we do not assume $\mathcal{T}_{\mathcal{I}}$ to be a binary tree, we get

$$C_{qr}k^3 \#\operatorname{sons}(\mathcal{T}_{\mathcal{I}}, t) \qquad and \qquad C_{qr}k^2 \#\operatorname{sons}(\mathcal{T}_{\mathcal{I}}, t)$$

operations, respectively, and we can conclude that preparing the matrices $\widehat{P}_t$ for the entire cluster tree takes not more than $C_{qr}k^3 \#\mathcal{T}_{\mathcal{I}}$ operations, while coarsening a hierarchical vector with a subtree $\mathcal{T}_x$ takes not more than $C_{qr}k^2 \#\mathcal{T}_x$ operations.

4 $\mathcal{H}^2$-matrices

We have developed algorithms for adding hierarchical vectors, for computing norms and inner products, and for refining and coarsening the corresponding subtrees.

Now we consider the multiplication of a hierarchical vector by a matrix. Let $\mathcal{T}_{\mathcal{I}}$ and $\mathcal{T}_{\mathcal{J}}$ be cluster trees for the index sets $\mathcal{I}$ and $\mathcal{J}$. We are looking for an algorithm that takes a matrix $A \in \mathbb{R}^{\mathcal{I} \times \mathcal{J}}$ and a hierarchical vector $x \in \mathbb{R}^{\mathcal{J}}$ corresponding to a subtree $\mathcal{T}_x$ of $\mathcal{T}_{\mathcal{J}}$ and computes a new hierarchical vector $y \in \mathbb{R}^{\mathcal{I}}$ such that $y = Ax$, and we would like this computation to take only $\mathcal{O}(k^2 \#\mathcal{T}_x)$ operations.

This is obviously not possible for general matrices A, so we have to restrict our attention to a suitable subset of matrices. $\mathcal{H}^2$-*matrices* [12, 5, 3] have the necessary properties.

Just like a hierarchical vector is based on a cluster tree $\mathcal{T}_{\mathcal{I}}$ that describes a hierarchical splitting of the index set $\mathcal{I}$, an $\mathcal{H}^2$-matrix is based on a *block tree* that describes a hierarchical splitting of $\mathcal{I} \times \mathcal{J}$.

Definition 4.1 (Block tree) *Let $\mathcal{T}_{\mathcal{I} \times \mathcal{J}} = (V, r, S, \iota)$ be a labeled tree. We call it a block tree for the cluster trees $\mathcal{T}_{\mathcal{I}}$ and $\mathcal{T}_{\mathcal{J}}$ if*

- *for each $b \in V$ there are $t \in \mathcal{T}_{\mathcal{I}}$ and $s \in \mathcal{T}_{\mathcal{J}}$ such that $b = (t, s)$ and $\hat{b} = \hat{t} \times \hat{s}$,*

- $\operatorname{root}(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}) = (\operatorname{root}(\mathcal{T}_{\mathcal{I}}), \operatorname{root}(\mathcal{T}_{\mathcal{J}}))$,

- *if $b = (t, s) \in V$ is not a leaf, we have $\operatorname{sons}(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}, b) = \operatorname{sons}(\mathcal{T}_{\mathcal{I}}, t) \times \operatorname{sons}(\mathcal{T}_{\mathcal{J}}, s)$.*

If $\mathcal{T}_{\mathcal{I} \times \mathcal{J}}$ is a block tree for $\mathcal{T}_{\mathcal{I}}$ and $\mathcal{T}_{\mathcal{J}}$, we call the elements $b \in V$ blocks and use the short notation $b \in \mathcal{T}_{\mathcal{I} \times \mathcal{J}}$ for $b \in V$. We denote its leaves by $\mathcal{L}_{\mathcal{I} \times \mathcal{J}}$.

It is easy to see that a block tree $\mathcal{T}_{\mathcal{I} \times \mathcal{J}}$ for $\mathcal{T}_{\mathcal{I}}$ and $\mathcal{T}_{\mathcal{J}}$ is a special cluster tree for the index set $\mathcal{I} \times \mathcal{J}$, and Remark 2.4 yields that

$$\{\hat{t} \times \hat{s} \ : \ b = (t, s) \in \mathcal{L}_{\mathcal{I} \times \mathcal{J}}\}$$

is a disjoint partition of $\mathcal{I} \times \mathcal{J}$, so we can describe a matrix $A \in \mathbb{R}^{\mathcal{I} \times \mathcal{J}}$ uniquely by defining its restrictions $A|_{\hat{t} \times \hat{s}}$ for all $b = (t, s) \in \mathcal{L}_{\mathcal{I} \times \mathcal{J}}$. An $\mathcal{H}^2$-matrix represents these submatrices by a three-term factorization using cluster bases.

Definition 4.2 ($\mathcal{H}^2$-matrix) *Let $\mathcal{T}_{\mathcal{I} \times \mathcal{J}}$ be a block tree for $\mathcal{T}_{\mathcal{I}}$ and $\mathcal{T}_{\mathcal{J}}$. Let $(V_t)_{t \in \mathcal{T}_{\mathcal{I}}}$ and $(W_s)_{s \in \mathcal{T}_{\mathcal{J}}}$ be cluster bases for $\mathcal{T}_{\mathcal{I}}$ and $\mathcal{T}_{\mathcal{J}}$, respectively.*

We call a matrix $A \in \mathbb{R}^{\mathcal{I} \times \mathcal{J}}$ an $\mathcal{H}^2$-matrix with respect to $\mathcal{T}_{\mathcal{I} \times \mathcal{J}}$, $(V_t)_{t \in \mathcal{T}_{\mathcal{I}}}$ and $(W_s)_{s \in \mathcal{T}_{\mathcal{J}}}$ if for each $b = (t, s) \in \mathcal{L}_{\mathcal{I} \times \mathcal{J}}$ we can find $S_b \in \mathbb{R}^{k \times k}$ such that

$$A|_{\hat{t} \times \hat{s}} = V_t S_b W_s^*. \tag{4.1}$$

In this case, the matrices S_b are called coupling matrices, *the cluster basis $(V_t)_{t \in \mathcal{T}_{\mathcal{I}}}$ is called the* row cluster basis, *and the cluster basis $(W_s)_{s \in \mathcal{T}_{\mathcal{J}}}$ is called the* column cluster basis.

Remark 4.3 (Special case) *A more general definition of $\mathcal{H}^2$-matrices is commonly found in the literature [12, 5, 3]. Our definition is equivalent if*

- *all leaves of the cluster trees $\mathcal{T}_{\mathcal{I}}$ and $\mathcal{T}_{\mathcal{J}}$ appear on the same level, and*

- *for all leaves $t \in \mathcal{L}_{\mathcal{I}}$ and $s \in \mathcal{L}_{\mathcal{J}}$, the matrices V_t and W_s have full rank.*

The first assumption allows us to avoid special cases in the construction of the block tree, the second assumption allows us to express all leaf blocks in the form (4.1), even if they do not satisfy the admissibility conditions *that are usually employed to determine approximability.*

We make both assumptions only to keep the presentation simple, all algorithms and theoretical arguments in the following can be extended to the general case by handling a moderate number of special cases.

5 Matrix-vector multiplication and induced bases

Let $x \in \mathbb{R}^{\mathcal{J}}$ be a hierarchical vector for a subtree $\mathcal{T}_x$ of $\mathcal{T}_{\mathcal{J}}$ and an isometric cluster basis $(Q_t)_{t \in \mathcal{T}_{\mathcal{I}}}$.

Let $A \in \mathbb{R}^{\mathcal{I} \times \mathcal{J}}$ be an $\mathcal{H}^2$-matrix for the block tree $\mathcal{T}_{\mathcal{I} \times \mathcal{J}}$, the row cluster basis $(V_t)_{t \in \mathcal{T}_{\mathcal{I}}}$ and the column cluster basis $(W_s)_{s \in \mathcal{T}_{\mathcal{J}}}$. We assume that both bases have rank k_A, while we keep using k to denote the rank used by the hierarchical vectors.

We want to compute the matrix-vector product $y := Ax \in \mathbb{R}^{\mathcal{I}}$ efficiently, i.e., the number of operations should be in $\mathcal{O}((k_A + k)^2 \mathcal{T}_x)$.

If x was a vector without hierarchical structure, we could split A recursively into submatrices according to the block tree and evaluate the contributions of the leaves to

the result. Since x is a hierarchical vector, we have to modify the procedure and stop splitting as soon as we reach a leaf of the subtree $\mathcal{T}_x$ describing the structure of x: let $b = (t, s) \in \mathcal{T}_{\mathcal{I} \times \mathcal{J}}$ with $s \in \mathcal{T}_x$. We consider the problem of evaluating $A|_{\hat{t} \times \hat{s}} x|_{\hat{s}}$.

1. If $\mathrm{sons}(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}, b) = \emptyset$, we have $A|_{\hat{t} \times \hat{s}} x|_{\hat{s}} = V_t S_b W_s^* x|_{\hat{s}}$ and can compute the product explicitly.

2. If $\mathrm{sons}(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}, b) \neq \emptyset$ and $\mathrm{sons}(\mathcal{T}_x, s) \neq \emptyset$, consider all submatrices $A|_{t' \times s'}$ with $b' = (t', s') \in \mathrm{sons}(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}, b)$ by recursion.

3. If $\mathrm{sons}(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}, b) \neq \emptyset$ and $\mathrm{sons}(\mathcal{T}_x, s) = \emptyset$, we have no choice but to compute $A|_{\hat{t} \times \hat{s}} x|_{\hat{s}}$ directly.

Case 2 is straightforward, we only have to ensure that the result y has a hierarchical structure that matches the recursion. This can be easily accomplished by using the procedure `refine`.

Case 1 can be handled as for standard $\mathcal{H}^2$-matrices: we prepare auxiliary vectors $\bar{x}_s = W_s^* x|_{\hat{s}}$ for all $s \in \mathcal{T}_x$ in advance by a *backward transformation*, accumulate all contributions to a row cluster $t \in \mathcal{T}_{\mathcal{I}}$ in an auxiliary vector $\bar{y}_t$, and finally add $V_t \bar{y}_t$ to the result using a *forward transformation*.

Let us first consider the backward transformation. If $s \in \mathcal{T}_x$ is a leaf, we have $x|_{\hat{s}} = Q_s \hat{x}_s$ by definition and need to compute

$$\bar{x}_s = W_s^* Q_s \hat{x}_s.$$

If we have the auxiliary matrices

$$D_s := W_s^* Q_s \in \mathbb{R}^{k_A \times k} \qquad \text{for all } s \in \mathcal{T}_{\mathcal{J}}$$

at our disposal, we can evaluate $\bar{x}_s = D_s \hat{x}_s$ in $\mathcal{O}(k_A k)$ operations. In order to prepare these matrices, we can follow a similar approach as in (2.4): denoting the transfer matrices for the column basis $(W_s)_{s \in \mathcal{T}_{\mathcal{J}}}$ by $(E_{W,s})_{s \in \mathcal{T}_{\mathcal{J}}}$ and the transfer matrices for the vector basis $(Q_s)_{s \in \mathcal{T}_{\mathcal{J}}}$ by $(F_s)_{s \in \mathcal{T}_{\mathcal{J}}}$, we can use (2.1) to get

$$D_s = \begin{cases} W_s^* Q_s & \text{if } \mathrm{sons}(\mathcal{T}_{\mathcal{J}}, s) = \emptyset, \\ \sum_{s' \in \mathrm{sons}(\mathcal{T}_{\mathcal{J}}, s)} E_{W,s'}^* D_{s'} F_{s'} & \text{otherwise} \end{cases} \qquad \text{for all } s \in \mathcal{T}_{\mathcal{J}}, \qquad (5.1)$$

and this allows us to compute all of these matrices in $\mathcal{O}(k_A(k_A + k)k \# \mathcal{T}_{\mathcal{J}})$ operations.

If $s \in \mathcal{T}_x$ is not a leaf, we can again use (2.1) to find

$$\bar{x}_s = W_s^* x|_{\hat{s}} = \sum_{s' \in \mathrm{sons}(\mathcal{T}_y, x)} W_s|_{\hat{s}' \times k}^* x|_{\hat{s}'} = \sum_{s' \in \mathrm{sons}(\mathcal{T}_y, x)} E_{W,s'}^* W_{s'}^* x|_{\hat{s}'}$$

$$= \sum_{s' \in \mathrm{sons}(\mathcal{T}_y, x)} E_{W,s'}^* \bar{x}_{s'} \qquad \text{for all } s \in \mathcal{T}_y, \ \mathrm{sons}(\mathcal{T}_y, s) \neq \emptyset.$$

The resulting algorithm is called the *forward transformation* and is given as the procedure `forward` in Figure 5.

```
procedure forward(s, x, var (x̄_s)_{s∈𝒯_x});          procedure backward(t, var (ȳ_t)_{t∈𝒯_y}, y);
if sons(𝒯_x, s) = ∅ then                              if sons(𝒯_y, t) = ∅ then
  x̄_s ← D_s x̂_s                                         ŷ_t ← ŷ_t + ȳ_t
else begin                                           else
  x̄_s ← 0;                                             for t' ∈ sons(𝒯_y, t) do begin
  for s' ∈ sons(𝒯_x, s) do begin                         ȳ_{t'} ← ȳ_{t'} + E_{V,t'} ȳ_t;
    forward(s', x, (x̄_s)_{s∈𝒯_x});                         backward(t', (ȳ_t)_{t∈𝒯_y}, y)
    x̄_s ← x̄_s + E*_{W,s'} x̄_{s'}                          end
  end
end
```

Figure 5: Perform forward and backward transformations for hierarchical vectors

If our recursive algorithm encounters a leaf $b = (t, s) \in \mathcal{L}_{\mathcal{I} \times \mathcal{J}}$, it looks up $\bar{x}_s = W_s^* x|_{\hat{s}}$ among the vectors prepared by the forward transformation, multiplies it by S_b, and adds it to an auxiliary vector $\bar{y}_t$ that collects all contributions to a row cluster t. As mentioned in the discussion of case 2, we assume that a subtree $\mathcal{T}_y$ is created by the recursive procedure that ensures $t \in \mathcal{T}_y$, so we only need the auxiliary vectors for these clusters.

In a last step, we have to take care of these temporary results, i.e., we have to add $V_t \bar{y}_t$ to the final result for all $t \in \mathcal{T}_y$. If we represent the result $y \in \mathbb{R}^{\mathcal{I}}$ as a hierarchical vector with the cluster basis $(V_t)_{t \in \mathcal{T}_{\mathcal{I}}}$, we can handle leaves $t \in \mathcal{L}_y$ directly by adding $\bar{y}_t$ to the corresponding coefficient vector $\hat{y}_t$. If $t \in \mathcal{T}_y \setminus \mathcal{L}_y$ is not a leaf, we can use (2.1) again to find

$$(V_t \bar{y}_t)|_{\hat{t}'} + V_{t'} \bar{y}_{t'} = V_t|_{\hat{t}' \times k} \bar{y}_t + V_{t'} \bar{y}_{t'} = V_{t'}(E_{V,t'} \bar{y}_t + \bar{y}_{t'}) \qquad \text{for all } t' \in \text{sons}(\mathcal{T}_y, t),$$

i.e., instead of adding $V_t \bar{y}_t$ to $y|_{\hat{t}}$ directly, we can also add $E_{V,t'} \bar{y}_t$ to $\bar{y}_{t'}$ and handle the son clusters $t' \in \text{sons}(\mathcal{T}_y, t)$ by recursion. The resulting algorithm is called the *backward transformation* and is given as the procedure backward in Figure 5.

While the cases 1 and 2 can be handled essentially as in the case of standard $\mathcal{H}^2$-matrices, the case 3 requires special treatment: if we encounter a leaf $s \in \mathcal{L}_x$ and a block $(t, s) \in \mathcal{T}_{\mathcal{I} \times \mathcal{J}}$ that is not a leaf of the block tree, we cannot afford to subdivide s further, since we are aiming for an algorithm with only $\mathcal{O}((k^2 + k_A^2) \# \mathcal{T}_x)$ operations. We have to find a way to add

$$A|_{\hat{t} \times \hat{s}} x|_{\hat{s}} = A|_{\hat{t} \times \hat{s}} Q_s \hat{x}_s$$

to the subvector $y|_{\hat{t}}$ of the result y.

We can face this challenge by using *induced cluster bases* [3, Section 7.8]: instead of representing the result y in the row cluster basis $(V_t)_{t \in \mathcal{T}_{\mathcal{I}}}$, we use a cluster basis that also contains the products $A|_{\hat{t} \times \hat{s}} Q_s \in \mathbb{R}^{\hat{t} \times k}$ for all $t \in \mathcal{T}_{\mathcal{I}}$ with $(t, s) \in \mathcal{T}_{\mathcal{I} \times \mathcal{J}} \setminus \mathcal{L}_{\mathcal{I} \times \mathcal{J}}$.

To define the induced cluster basis, we introduce

$$\text{row}^-(t) := \{s \in \mathcal{T}_{\mathcal{J}} \ : \ (t, s) \in \mathcal{T}_{\mathcal{I} \times \mathcal{J}} \setminus \mathcal{L}_{\mathcal{I} \times \mathcal{J}}\} \qquad \text{for all } t \in \mathcal{T}_{\mathcal{I}}. \qquad (5.2)$$

This set contains all column clusters $s \in \mathcal{T}_{\mathcal{J}}$ that appear in non-leaf blocks with the row cluster t. These are the blocks that have to be handled by the induced basis. We let $\beta_t := \# \operatorname{row}^-(t)$ and fix $s_{t,1}, \ldots, s_{t,\beta_t} \in \mathcal{T}_{\mathcal{J}}$ such that

$$\operatorname{row}^-(t) = \{s_{t,1}, \ldots, s_{t,\beta_t}\} \qquad\qquad \text{for all } t \in \mathcal{T}_{\mathcal{I}}.$$

Definition 5.1 (Induced cluster basis) *Let $A \in \mathbb{R}^{\mathcal{I} \times \mathcal{J}}$ be an $\mathcal{H}^2$-matrix for $\mathcal{T}_{\mathcal{I} \times \mathcal{J}}$ with row cluster basis $(V_t)_{t \in \mathcal{T}_{\mathcal{I}}}$, and let $(Q_s)_{s \in \mathcal{T}_{\mathcal{J}}}$ be another cluster basis.*
We define $\ell_t := k_A + k\beta_t$ and

$$U_t := \begin{pmatrix} V_t & A|_{\hat{t} \times \hat{s}_{t,1}} Q_{s_{t,1}} & \cdots & A|_{\hat{t} \times \hat{s}_{t,\beta_t}} Q_{s_{t,\beta_t}} \end{pmatrix} \in \mathbb{R}^{\hat{t} \times \ell_t} \qquad \text{for all } t \in \mathcal{T}_{\mathcal{I}} \qquad (5.3)$$

and call $(U_t)_{t \in \mathcal{T}_{\mathcal{I}}}$ the induced cluster basis corresponding to the $\mathcal{H}^2$-matrix A and the input cluster basis $(Q_s)_{s \in \mathcal{T}_{\mathcal{J}}}$.

Remark 5.2 (Nested) *Calling the induced cluster basis $(U_t)_{t \in \mathcal{T}_{\mathcal{I}}}$ a cluster basis is justified, since it is nested [3, Lemma 7.22], i.e., it satisfies (2.1) for suitable transfer matrices $E_{U,t'} \in \mathbb{R}^{\ell_{t'} \times \ell_t}$.*

If we use the induced cluster basis to represent the result y, case 3 can be handled by simply adding $\hat{x}_s$ to the appropriate portion of the corresponding coefficient vector. To keep the notation simple, we denote these coefficient vectors by

$$\hat{y}_{U,t} = \begin{pmatrix} \hat{y}_t \\ \hat{y}_{t,s_{t,1}} \\ \vdots \\ \hat{y}_{t,s_{t,\beta_t}} \end{pmatrix} \in \mathbb{R}^{\ell_t}, \qquad\qquad \bar{y}_{U,t} = \begin{pmatrix} \bar{y}_t \\ \bar{y}_{t,s_{t,1}} \\ \vdots \\ \bar{y}_{t,s_{t,\beta_t}} \end{pmatrix} \in \mathbb{R}^{\ell_t}.$$

Now the cases 1 and 3 can be handled almost in the same way. Figure 6 summarizes the recursive procedure `coupling` that handles all three cases.

To complete our algorithm for the matrix-vector multiplication, we require the backward transformation for the induced cluster basis. In order to generalize the algorithm `backward` given in Figure 5, we require an understanding of how the transfer matrices for the induced cluster basis act on vectors.

In order to keep the presentation simple, we again assume that the cluster tree $\mathcal{T}_{\mathcal{J}}$ is binary and that for each cluster $s \in \mathcal{T}_{\mathcal{J}} \setminus \mathcal{L}_{\mathcal{J}}$ its sons are given by $\operatorname{sons}(\mathcal{T}_{\mathcal{J}}, s) = \{s_1, s_2\}$.

The first block of (5.3) is straightforward: for $t \in \mathcal{T}_{\mathcal{I}}$ and $t' \in \operatorname{sons}(\mathcal{T}_{\mathcal{I}}, t)$, we have

$$V_t|_{\hat{t}' \times k} = V_{t'} E_{t'}$$

by (2.1). The following blocks are a little more involved. Let $s \in \operatorname{row}^-(t)$. The definition (5.2) yields $b := (t,s) \in \mathcal{T}_{\mathcal{I} \times \mathcal{J}} \setminus \mathcal{L}_{\mathcal{I} \times \mathcal{J}}$, and with Definition 4.1 we obtain $\operatorname{sons}(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}, b) = \operatorname{sons}(t) \times \operatorname{sons}(s)$. Restricting to t' and using (2.1) gives us

$$(A|_{\hat{t} \times \hat{s}} Q_s)|_{\hat{t}' \times k} = A|_{\hat{t}' \times \hat{s}} Q_s = \begin{pmatrix} A|_{\hat{t}' \times \hat{s}_1} & A|_{\hat{t}' \times \hat{s}_2} \end{pmatrix} \begin{pmatrix} Q_{s_1} F_{s_1} \\ Q_{s_2} F_{s_2} \end{pmatrix}$$

$$= \begin{pmatrix} A|_{\hat{t}' \times \hat{s}_1} Q_{s_1} & A|_{\hat{t}' \times \hat{s}_2} Q_{s_2} \end{pmatrix} \begin{pmatrix} F_{s_1} \\ F_{s_2} \end{pmatrix}.$$

procedure `coupling`$(b = (t, s), x, (\bar{x}_s)_{s \in \mathcal{T}_x}, \textbf{var } y, (\bar{y}_t)_{t \in \mathcal{T}_y})$;
if $\text{sons}(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}, b) = \emptyset$ **then**
$\quad \bar{y}_t \leftarrow \bar{y}_t + S_b \bar{x}_s$
else if $\text{sons}(\mathcal{T}_x, s) = \emptyset$ **then**
$\quad \bar{y}_{t,s} \leftarrow \bar{y}_{t,s} + \hat{x}_s$
else begin
$\quad$ **if** $\text{sons}(\mathcal{T}_y, t) = \emptyset$ **then begin**
$\quad\quad$ `refine`(t, y);
$\quad\quad$ **for** $t' \in \text{sons}(\mathcal{T}_{\mathcal{I}}, t)$ **do** $\bar{y}_{U,t'} \leftarrow 0$
$\quad$ **end**;
$\quad$ **for** $t' \in \text{sons}(\mathcal{T}_{\mathcal{I}}, t), s' \in \text{sons}(\mathcal{T}_{\mathcal{J}}, s)$ **do**
$\quad\quad$ `coupling`$(b' = (t', s'), x, (\bar{x}_s)_{s \in \mathcal{T}_x}, y, (\bar{y}_t)_{t \in \mathcal{T}_y})$
end

Figure 6: Evaluating all couplings between row and column clusters

Let $s' \in \text{sons}(\mathcal{T}_{\mathcal{J}}, s)$. If $b' := (t', s')$ is a leaf of $\mathcal{T}_{\mathcal{I} \times \mathcal{J}}$, we have

$$A|_{\hat{t}' \times \hat{s}'} Q_{s'} F_{s'} \bar{y}_{t,s} = V_{t'} S_{b'} W_{s'}^* Q_{s'} F_{s'} \bar{y}_{t,s} = V_{t'} S_{b'} D_{s'} F_{s'} \bar{y}_{t,s},$$

and we can express the product by the first block in (5.3).

On the other hand, if $b' := (t', s')$ is not a leaf of $\mathcal{T}_{\mathcal{I} \times \mathcal{J}}$, we have $s' \in \text{row}^-(t')$ by definition and can express the product $A|_{\hat{t}' \times \hat{s}'} Q_{s'} F_{s'}$ using one of the other blocks in (5.3). The resulting backward transformation for the induced cluster basis is summarized as the procedure `induced_backward` in Figure 7.

Combining the forward transformation given in Figure 5, the coupling step in Figure 6, and the backward transformation for the induced basis in Figure 7 yields the matrix-vector multiplication algorithm given in Figure 8.

Our goal is now to prove that this algorithm requires not more than $\mathcal{O}((k_A^2 + k^2) \# \mathcal{T}_x)$ operations. In order to establish a connection between the number of clusters and the number of blocks, we require a standard assumption: the block tree has to be *sparse* [11, 10].

Definition 5.3 (Sparse) *Let $\mathcal{T}_{\mathcal{I} \times \mathcal{J}}$ be a block tree for $\mathcal{T}_{\mathcal{I}}$ and $\mathcal{T}_{\mathcal{J}}$. We define*

$$\begin{aligned} \text{row}(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}, t) &:= \{s \in \mathcal{T}_{\mathcal{J}} : (t, s) \in \mathcal{T}_{\mathcal{I} \times \mathcal{J}}\} && \textit{for all } t \in \mathcal{T}_{\mathcal{I}}, \\ \text{col}(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}, s) &:= \{t \in \mathcal{T}_{\mathcal{I}} : (t, s) \in \mathcal{T}_{\mathcal{I} \times \mathcal{J}}\} && \textit{for all } s \in \mathcal{T}_{\mathcal{J}}. \end{aligned}$$

Let $C_{sp} \in \mathbb{N}$. A block tree $\mathcal{T}_{\mathcal{I} \times \mathcal{J}}$ is called C_{sp}-sparse if

$$\# \text{row}(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}, t) \leq C_{sp}, \qquad \# \text{col}(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}, s) \leq C_{sp} \qquad \textit{for all } t \in \mathcal{T}_{\mathcal{I}}, \ s \in \mathcal{T}_{\mathcal{J}}.$$

Lemma 5.4 (Forward transformation) *The forward transformation `forward` given in Figure 5 requires $\mathcal{O}(k_A(k_A + k) \# \mathcal{T}_x)$ operations.*

procedure induced_backward(t, **var** $(\bar{y}_{U,t})_{t \in \mathcal{T}_y}$, y);
if $\mathrm{sons}(\mathcal{T}_y, t) = \emptyset$ **then**
   $\hat{y}_{U,t} \leftarrow \hat{y}_{U,t} + \bar{y}_{U,t}$
else
  for $t' \in \mathrm{sons}(\mathcal{T}_y, t)$ **do begin**
    $\bar{y}_{t'} \leftarrow \bar{y}_{t'} + E_{V,t'}\bar{y}_t$;
    for $s \in \mathrm{row}^-(t)$, $s' \in \mathrm{sons}(\mathcal{T}_{\mathcal{J}}, s)$ **do begin**
      $b' \leftarrow (t', s')$;
      if $\mathrm{sons}(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}, b') = \emptyset$ **then**
        $\bar{y}_{t'} \leftarrow \bar{y}_{t'} + S_{b'}D_{s'}F_{s'}\bar{y}_{t,s}$
      else
        $\bar{y}_{t',s'} \leftarrow \bar{y}_{t',s'} + F_{s'}\bar{y}_{t,s}$
    end;
    induced_backward(t', $(\bar{y}_{U,t})_{t \in \mathcal{T}_y}$, y)
  end

Figure 7: Backward transformation for the induced cluster basis

procedure eval(A, x, **var** y);
$r_{\mathcal{I}} \leftarrow \mathrm{root}(\mathcal{T}_{\mathcal{I}})$; $r_{\mathcal{J}} \leftarrow \mathrm{root}(\mathcal{T}_{\mathcal{J}})$;
Let $\mathcal{T}_y$ be the minimal subtree of $\mathcal{T}_{\mathcal{I}}$ containing only $r_{\mathcal{I}}$;
$\bar{y}_{U,r_{\mathcal{I}}} \leftarrow 0$; $\hat{y}_{U,r_{\mathcal{I}}} \leftarrow 0$;
forward($r_{\mathcal{J}}$, x, $(\bar{x}_s)_{s \in \mathcal{T}_x}$);
coupling($b = (r_{\mathcal{I}}, r_{\mathcal{J}})$, x, $(\bar{x}_s)_{s \in \mathcal{T}_x}$, y, $(\bar{y}_t)_{t \in \mathcal{T}_y}$);
induced_backward($r_{\mathcal{I}}$, $(\bar{y}_{U,t})_{t \in \mathcal{T}_y}$, y)

Figure 8: Matrix-vector multiplication, result represented in the induced cluster basis

Proof. We first note that the function `forward` is only called for clusters $s \in \mathcal{T}_x$.

If s is a leaf, the multiplication by D_s requires $2k_A k$ operations.

If s is not a leaf, the function performs a multiplication by $E_{s'}$ for each of the sons $s' \in \text{sons}(\mathcal{T}_x, s)$. This takes $2k_A^2$ operations.

Since each cluster has at most one father, not more than $2k_A(k_A + k)$ operations are required for each cluster. ∎

Lemma 5.5 (Coupling step) *Let $\mathcal{T}_{\mathcal{I} \times \mathcal{J}}$ be C_{sp}-sparse.*

The coupling step `coupling` *given in Figure 6 requires $\mathcal{O}(C_{sp} k_A(k_A + k) \# \mathcal{T}_x)$ operations.*

If $\mathcal{T}_y$ is the minimal subtree of $\mathcal{T}_{\mathcal{I}}$ containing only the root prior to calling `coupling`, *it will satisfy*

$$\# \mathcal{T}_y \leq C_{sp} \# \mathcal{T}_x$$

after completion of the algorithm.

Proof. The procedure `coupling` is only called recursively if $\text{sons}(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}, b) \neq \emptyset$ and $\text{sons}(\mathcal{T}_x, s) \neq \emptyset$ hold. In this case, we have $\text{sons}(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}, b) = \text{sons}(\mathcal{T}_{\mathcal{I}}, t) \times \text{sons}(\mathcal{T}_{\mathcal{J}}, s)$ and $\text{sons}(\mathcal{T}_x, s) = \text{sons}(\mathcal{T}_{\mathcal{J}}, s)$ by definition, and therefore $(t', s') \in \mathcal{T}_{\mathcal{I} \times \mathcal{J}}$ and $s' \in \mathcal{T}_y$ for all $t' \in \text{sons}(\mathcal{T}_{\mathcal{I}})$ and $s' \in \text{sons}(\mathcal{T}_{\mathcal{J}})$. Since `coupling` is first called with $t = \text{root}(\mathcal{T}_{\mathcal{I}})$ and $s = \text{root}(\mathcal{T}_{\mathcal{J}})$, we can guarantee that for each call to `coupling` we have $b = (t, s) \in \mathcal{T}_{\mathcal{I} \times \mathcal{J}}$ and $s \in \mathcal{T}_x$.

This implies $t \in \text{col}(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}, s)$, and since the block tree $\mathcal{T}_{\mathcal{I} \times \mathcal{J}}$ is C_{sp}-sparse, we have $\# \text{col}(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}, s) \leq C_{sp}$.

In each call to `coupling`, we perform either $2k_A^2$ operations to multiply $\bar{x}_s$ by S_b, or $k \leq 2k_A k$ operations to add $\hat{x}_s$ to $\bar{y}_{t,s}$.

We conclude that the total number of arithmetic operations is bounded by

$$\sum_{s \in \mathcal{T}_x} \sum_{t \in \text{col}(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}, s)} 2k_A(k_A + k) \leq \sum_{s \in \mathcal{T}_x} 2C_{sp} k_A(k_A + k) = 2C_{sp} k_A(k_A + k) \# \mathcal{T}_x.$$

The procedure `refine` is only called to extend the tree $\mathcal{T}_y$ if $\text{sons}(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}, b) \neq \emptyset$ and $\text{sons}(\mathcal{T}_x, s) \neq \emptyset$. We have already seen that in this case we have $(t', s') \in \mathcal{T}_{\mathcal{I} \times \mathcal{J}}$ and $s' \in \mathcal{T}_x$ for all $t' \in \text{sons}(\mathcal{T}_{\mathcal{I}}, t)$ and $s' \in \text{sons}(\mathcal{T}_{\mathcal{J}}, s)$. In particular, for each $t' \in \text{sons}(\mathcal{T}_{\mathcal{I}}, t)$ added by `refine`, we can find a cluster $s' \in \text{sons}(\mathcal{T}_x, s) \subseteq \mathcal{T}_x$ with $t' \in \text{col}(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}, s')$.

Since we start the procedure with a minimal subtree $\mathcal{T}_y$ containing only the root of $\mathcal{T}_{\mathcal{I}}$, we can conclude that $t \in \mathcal{T}_y$ implies $t \in \text{col}(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}, s)$ for an $s \in \mathcal{T}_x$. The C_{sp}-sparsity of $\mathcal{T}_{\mathcal{I} \times \mathcal{J}}$ yields

$$\mathcal{T}_y \subseteq \bigcup_{s \in \mathcal{T}_x} \text{col}(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}, s), \qquad \# \mathcal{T}_y \leq \sum_{s \in \mathcal{T}_x} \# \text{col}(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}, s) \leq C_{sp} \# \mathcal{T}_x.$$

∎

Lemma 5.6 (Backward transformation) *Let $\mathcal{T}_{\mathcal{I} \times \mathcal{J}}$ be C_{sp}-sparse.*
The ranks of the induced cluster basis are bounded by

$$\ell_t \leq k_A + C_{sp}k \qquad\qquad \text{for all } t \in \mathcal{T}_{\mathcal{I}},$$

and the backward transformation `induced_backward` *for it given in Figure 7 requires*
$\mathcal{O}(C_{sp}(k_A^2 + k^2)\#\mathcal{T}_y)$ *operations.*

Proof. We first note that we have

$$\#\operatorname{row}^-(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}, t) \leq \#\operatorname{row}(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}, t) \leq C_{\mathrm{sp}} \qquad\qquad \text{for all } t \in \mathcal{T}_{\mathcal{I}}.$$

By definition (5.3), this implies

$$\ell_t \leq k_A + C_{\mathrm{sp}}k \qquad\qquad \text{for all } t \in \mathcal{T}_{\mathcal{I}}.$$

Let us now consider the number of operations required by `induced_backward` for a cluster $t \in \mathcal{T}_y$.

If t is a leaf, $\bar{y}_{U,t}$ is added to $\hat{y}_{U,t}$, and due to (5.3), this requires

$$k_A + k\#\operatorname{row}^-(t) \leq k_A + k\#\operatorname{row}(t)$$

operations.

If t is not a leaf, the multiplication of $\bar{y}_t$ with $E_{V,t'}$ takes $2k_A^2$ operations, and for all $s \in \operatorname{row}^-(t)$ and $s' \in \operatorname{sons}(\mathcal{T}_{\mathcal{J}}, s)$ we perform either one multiplication with $F_{s'}$ or three multiplications with $S_{b'}$, $D_{s'}$, and $F_{s'}$, so not more than $2k_A^2 + 2k_Ak + 2k^2 \leq 3(k_A^2 + k^2)$ operations are required. Due to $t' \in \operatorname{sons}(\mathcal{T}_y, t)$, $s \in \operatorname{row}^-(t)$ and $s' \in \operatorname{sons}(\mathcal{T}_{\mathcal{J}}, s)$, we have $(t', s') \in \mathcal{T}_{\mathcal{I} \times \mathcal{J}}$ and therefore $s' \in \operatorname{row}(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}, t')$. Since each s' has only one father s, we conclude that not more than

$$2k_A^2 + \sum_{s \in \operatorname{row}^-(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}, t)} \sum_{s' \in \operatorname{sons}(\mathcal{T}_{\mathcal{J}}, s)} 3(k_A^2 + k^2) = 2k_A^2 + \sum_{s' \in \operatorname{row}(\mathcal{T}_{\mathcal{I} \times \mathcal{J}}, t')} 3(k_A^2 + k^2)$$
$$= 2k_A^2 + 3(k_A^2 + k^2)\#\operatorname{row}(t')$$

operations are required. The total number of operations is now bounded by

$$\sum_{t \in \mathcal{T}_y} 2k_A^2 + 3(k_A^2 + k^2)\#\operatorname{row}(t) \leq \sum_{t \in \mathcal{T}_y} 2k_A^2 + 3(k_A^2 + k^2)C_{\mathrm{sp}}$$
$$\leq (2 + 3C_{\mathrm{sp}})(k_A^2 + k^2)\#\mathcal{T}_y.$$

∎

Theorem 5.7 (Complexity, matrix-vector multiplication) *Let the blocktree $\mathcal{T}_{\mathcal{I} \times \mathcal{J}}$ be C_{sp}-sparse.*
The algorithm `eval` *given in Figure 8 requires* $\mathcal{O}(C_{sp}^2(k_A^2 + k^2)\#\mathcal{T}_x)$ *operations.*

Proof. Lemma 5.4 yields that the forward transformation requires $\mathcal{O}(k_A(k_A + k)\#\mathcal{T}_x) \subseteq \mathcal{O}((k_A^2 + k^2)\#\mathcal{T}_x)$ operations.

Lemma 5.5 yields that the coupling step requires $\mathcal{O}(C_{\mathrm{sp}}k_A(k_A+k)\#\mathcal{T}_x) \subseteq \mathcal{O}(C_{\mathrm{sp}}(k_A^2 + k^2)\#\mathcal{T}_x)$ operations and that $\mathcal{T}_y$ will subsequently satisfy $\#\mathcal{T}_y \leq C_{\mathrm{sp}}\#\mathcal{T}_x$.

Lemma 5.6 yields that the backward transformation requires $\mathcal{O}(C_{\mathrm{sp}}(k_A^2 + k^2)\#\mathcal{T}_y)$ operations, and due to the estimate $\#\mathcal{T}_y \leq C_{\mathrm{sp}}\#\mathcal{T}_x$, the number of operations is also in $\mathcal{O}(C_{\mathrm{sp}}^2(k_A^2 + k^2)\#\mathcal{T}_x)$.

Setting up the initial vector y requires no arithmetic operations. ∎

6 Adaptive conversion

We have seen that the matrix-vector multiplication algorithm presented in Figures 5, 6, 7 and 8 computes the exact result of the matrix vector multiplication $y = Ax$ for a hierarchical vector x in $\mathcal{O}(C_{\mathrm{sp}}^2(k_A^2 + k^2)\#\mathcal{T}_x)$ operations.

Unfortunately, the algorithm yields a result that does not use a cluster basis of our choosing, but the somewhat artificial induced cluster basis. This is particularly undesirable since the rank of the induced cluster basis can become quite large.

We address this problem by developing an algorithm that approximates a given hierarchical vector by another hierarchical vector using a different basis.

Let $x \in \mathbb{R}^{\mathcal{I}}$ be a hierarchical vector corresponding to a subtree $\mathcal{T}_x$ and a cluster basis $(V_t)_{t \in \mathcal{T}_{\mathcal{I}}}$. Our goal is to represent it by a hierarchical vector $y \in \mathbb{R}^{\mathcal{I}}$ corresponding to a subtree $\mathcal{T}_y$ and a second cluster basis $(Q_t)_{t \in \mathcal{T}_{\mathcal{I}}}$. We have already seen that isometric cluster bases are useful for purposes like this, so we assume that $(Q_t)_{t \in \mathcal{T}_{\mathcal{I}}}$ is isometric.

Consider a leaf $t \in \mathcal{L}_x$. We have $x|_{\hat{t}} = V_t \hat{x}_t$, and we could simply employ the orthogonal projection corresponding to Q_t to obtain the approximation

$$x|_{\hat{t}} = V_t \hat{x}_t \approx Q_t Q_t^* V_t \hat{x}_t,$$

but we are faced with the question if this approximation is sufficiently accurate.

Assume that we know that it is not. In this case, we split t into its sons and try to approximate the subvectors $x|_{\hat{t}'}$ by the projections $Q_{t'}Q_{t'}^* x|_{\hat{t}'}$ corresponding to the sons $t' \in \mathrm{sons}(\mathcal{T}_{\mathcal{I}}, t)$. If the resulting errors are still too large, we keep splitting recursively until we are satisfied. Since we have (2.1) at our disposal, this procedure can be carried out efficiently and provides us with the required hierarchical vector y and subtree $\mathcal{T}_y$.

Although $\mathcal{T}_y$ is now guaranteed to be fine enough to satisfy our accuracy requirements, it may be *too* fine. Fortunately, we have the procedure `coarsen` (cf. Figure 4) at our disposal to reduce the cluster tree again while ensuring that the error stays below a given bound.

The resulting algorithm `convert` is given in Figure 9, but it is still incomplete: how can the algorithm judge whether an approximation error is "small enough"?

For the second case, i.e., the coarsening of the vector y given in the isometric basis $(Q_t)_{t \in \mathcal{T}_{\mathcal{I}}}$, we have already solved this problem: we can construct the auxiliary matrices $(P_t)_{t \in \mathcal{T}_{\mathcal{I}}}$ introduced in (3.4) and use (3.5) to compute the error norm explicitly and robustly.

procedure `convert_leaf`(t, $\hat{x}_t$, **var** y);
if $\|V_t\hat{x}_t - Q_tQ_t^*V_t\hat{x}_t\|$ small enough **then**
  $\hat{y}_t \leftarrow Q_t^*V_t\hat{x}_t$
else begin
  `refine`(t, y);
  for $t' \in \mathrm{sons}(\mathcal{T}_{\mathcal{I}}, t)$ **do begin**
    $\hat{x}_{t'} \leftarrow E_{t'}\hat{x}_t$;
    `convert_leaf`(t', $\hat{x}_{t'}$, y)
  end
end

procedure `convert`(t, x, **var** y);
if $\mathrm{sons}(\mathcal{T}_x, t) = \emptyset$ **then**
  `convert_leaf`(t, $\hat{x}_t$, y)
end else begin
  for $t' \in \mathrm{sons}(\mathcal{T}_x, t)$ **do**
    `convert`(t', x, y);
  if $\||y|_{\hat{t}} - Q_tQ_t^*y|_{\hat{t}}\|$ small enough **then**
    `coarsen`(t, y)
end

Figure 9: Conversion from one cluster basis to another with adaptively constructed subtree

We still have to consider the first case: a leaf $t \in \mathcal{T}_x$ is given and we have to compute the projection error

$$\|V_t\hat{x}_t - Q_tQ_t^*V_t\hat{x}_t\|.$$

We can solve this problem by generalizing the approach used for the coarsening error: we construct $k \times k$ matrices $(Z_t)_{t \in \mathcal{T}_{\mathcal{I}}}$ and isometric matrices $(P_t)_{t \in \mathcal{T}_{\mathcal{I}}}$ such that

$$
\begin{aligned}
V_t - Q_tQ_t^*V_t &= P_tZ_t & \text{for all } t \in \mathcal{T}_{\mathcal{I}}, & \qquad (6.1\text{a}) \\
P_t^*P_t &= I & \text{for all } t \in \mathcal{T}_{\mathcal{I}}, & \qquad (6.1\text{b}) \\
P_t^*Q_t &= 0 & \text{for all } t \in \mathcal{T}_{\mathcal{I}}. & \qquad (6.1\text{c})
\end{aligned}
$$

The first property (6.1a) states that P_tZ_t is a representation of the projection error, the second property (6.1b) simply restates that P_t is isometric. The third property (6.1c) is used in the construction of the families $(Z_t)_{t \in \mathcal{T}_{\mathcal{I}}}$ and $(P_t)_{t \in \mathcal{T}_{\mathcal{I}}}$.

Combining (6.1a) and (6.1b) yields

$$\|V_t\hat{x}_t - Q_tQ_t^*V_t\hat{x}_t\|_2 = \|P_tZ_t\hat{x}_t\|_2 = \|Z_t\hat{x}_t\|_2 \qquad \text{for all } t \in \mathcal{T}_{\mathcal{I}}, \ \hat{x}_t \in \mathbb{R}^k,$$

and since Z_t is small, we can evaluate the right-hand side efficiently.

We construct the families $(Z_t)_{t \in \mathcal{T}_{\mathcal{I}}}$ and $(P_t)_{t \in \mathcal{T}_{\mathcal{I}}}$ by recursion. Let $t \in \mathcal{T}_{\mathcal{I}}$.

Leaf cluster. If t is a leaf of $\mathcal{T}_{\mathcal{I}}$, we have V_t and Q_t at our disposal. We can use a Householder factorization to extend Q_t to an isometric matrix, i.e., to find an isometric matrix $\widetilde{P}_t$ such that

$$\widetilde{Q}_t := \begin{pmatrix} Q_t & \widetilde{P}_t \end{pmatrix} \in \mathbb{R}^{\hat{t} \times \hat{t}}$$

is orthogonal, i.e., quadratic and isometric. This means

$$V_t = \widetilde{Q}_t\widetilde{Q}_t^*V_t = \begin{pmatrix} Q_t & \widetilde{P}_t \end{pmatrix} \begin{pmatrix} Q_t^* \\ \widetilde{P}_t^* \end{pmatrix} V_t = Q_tQ_t^*V_t + \widetilde{P}_t\widetilde{P}_t^*V_t,$$

22

which is equivalent to

$$V_t - Q_t Q_t^* V_t = \widetilde{P}_t \widetilde{P}_t^* V_t.$$

If $k \ll \#\hat{t}$, the matrix $\widetilde{P}_t^* V_t$ will have a large number of rows. Since it has only k columns, we use a thin Householder factorization

$$\widetilde{P}_t^* V_t = \widehat{P}_t Z_t, \tag{6.2}$$

where Z_t is a $k \times k$ upper triangular matrix and $\widehat{P}_t$ is isometric. We let

$$P_t := \widetilde{P}_t \widehat{P}_t$$

and observe

$$V_t = Q_t Q_t^* V_t + \widetilde{P}_t \widetilde{P}_t^* V_t = Q_t Q_t^* V_t + \widetilde{P}_t \widehat{P}_t Z_t = Q_t Q_t^* V_t + P_t Z_t,$$
$$P_t^* P_t = \widehat{P}_t^* \widetilde{P}_t^* \widetilde{P}_t \widehat{P}_t = \widehat{P}_t^* \widehat{P}_t = I,$$
$$P_t^* Q_t = \widehat{P}_t^* \widetilde{P}_t^* Q_t = \widehat{P}_t^* 0 = 0.$$

Non-leaf cluster. If t is not a leaf of $\mathcal{T}_\mathcal{I}$, we cannot use V_t and Q_t, since they are only given implicitly via the corresponding transfer matrices. For the sake of simplicity, we assume $\#\operatorname{sons}(t) = 2$ and $\operatorname{sons}(t) = \{t_1, t_2\}$. We have

$$V_t = \begin{pmatrix} V_{t_1} E_{t_1} \\ V_{t_2} E_{t_2} \end{pmatrix}, \qquad\qquad Q_t = \begin{pmatrix} Q_{t_1} F_{t_1} \\ Q_{t_2} F_{t_2} \end{pmatrix} = \begin{pmatrix} Q_{t_1} & \\ & Q_{t_2} \end{pmatrix} \widehat{Q}_t,$$

where we let

$$\widehat{Q}_t := \begin{pmatrix} F_{t_1} \\ F_{t_2} \end{pmatrix}.$$

Assuming that P_{t_1}, Z_{t_1}, P_{t_2} and Z_{t_2} have already been computed by recursion, we find

$$\begin{aligned}
V_t - Q_t Q_t^* V_t &= \begin{pmatrix} V_{t_1} E_{t_1} \\ V_{t_2} E_{t_2} \end{pmatrix} - \begin{pmatrix} Q_{t_1} & \\ & Q_{t_2} \end{pmatrix} \widehat{Q}_t \widehat{Q}_t^* \begin{pmatrix} Q_{t_1}^* & \\ & Q_{t_2}^* \end{pmatrix} \begin{pmatrix} V_{t_1} E_{t_1} \\ V_{t_2} E_{t_2} \end{pmatrix} \\
&= \begin{pmatrix} V_{t_1} E_{t_1} \\ V_{t_2} E_{t_2} \end{pmatrix} - \begin{pmatrix} Q_{t_1} & \\ & Q_{t_2} \end{pmatrix} \widehat{Q}_t \widehat{Q}_t^* \begin{pmatrix} Q_{t_1}^* V_{t_1} E_{t_1} \\ Q_{t_2}^* V_{t_2} E_{t_2} \end{pmatrix} \\
&= \begin{pmatrix} V_{t_1} E_{t_1} \\ V_{t_2} E_{t_2} \end{pmatrix} - \begin{pmatrix} Q_{t_1} Q_{t_1}^* V_{t_1} E_{t_1} \\ Q_{t_2} Q_{t_2}^* V_{t_2} E_{t_2} \end{pmatrix} \\
&\quad + \begin{pmatrix} Q_{t_1} Q_{t_1}^* V_{t_1} E_{t_1} \\ Q_{t_2} Q_{t_2}^* V_{t_2} E_{t_2} \end{pmatrix} - \begin{pmatrix} Q_{t_1} & \\ & Q_{t_2} \end{pmatrix} \widehat{Q}_t \widehat{Q}_t^* \begin{pmatrix} Q_{t_1}^* V_{t_1} E_{t_1} \\ Q_{t_2}^* V_{t_2} E_{t_2} \end{pmatrix} \\
&= \begin{pmatrix} (V_{t_1} - Q_{t_1} Q_{t_1}^* V_{t_1}) E_{t_1} \\ (V_{t_2} - Q_{t_2} Q_{t_2}^* V_{t_2}) E_{t_2} \end{pmatrix} \\
&\quad + \begin{pmatrix} Q_{t_1} & \\ & Q_{t_2} \end{pmatrix} \left[\begin{pmatrix} Q_{t_1}^* V_{t_1} E_{t_1} \\ Q_{t_2}^* V_{t_2} E_{t_2} \end{pmatrix} - \widehat{Q}_t \widehat{Q}_t^* \begin{pmatrix} Q_{t_1}^* V_{t_1} E_{t_1} \\ Q_{t_2}^* V_{t_2} E_{t_2} \end{pmatrix} \right].
\end{aligned}$$

For the first term, our assumption yields

$$\begin{pmatrix} (V_{t_1} - Q_{t_1} Q_{t_1}^* V_{t_1}) E_{t_1} \\ (V_{t_2} - Q_{t_2} Q_{t_2}^* V_{t_2}) E_{t_2} \end{pmatrix} = \begin{pmatrix} P_{t_1} Z_{t_1} E_{t_1} \\ P_{t_2} Z_{t_2} E_{t_2} \end{pmatrix}. \tag{6.3}$$

For the second term, we introduce

$$\widehat{V}_t := \begin{pmatrix} Q_{t_1}^* V_{t_1} E_{t_1} \\ Q_{t_2}^* V_{t_2} E_{t_2} \end{pmatrix}$$

and obtain

$$\begin{pmatrix} Q_{t_1}^* V_{t_1}^* E_{t_1} \\ Q_{t_2}^* V_{t_2}^* E_{t_2} \end{pmatrix} - \widehat{Q}_t \widehat{Q}_t^* \begin{pmatrix} Q_{t_1}^* V_{t_1}^* E_{t_1} \\ Q_{t_2}^* V_{t_2}^* E_{t_2} \end{pmatrix} = \widehat{V}_t - \widehat{Q}_t \widehat{Q}_t^* \widehat{V}_t.$$

As before, we extend $\widehat{Q}_t$ to a square isometric matrix

$$\widetilde{Q}_t := \begin{pmatrix} \widehat{Q}_t & \widetilde{P}_t \end{pmatrix}$$

and obtain

$$\widehat{V}_t - \widehat{Q}_t \widehat{Q}_t^* \widehat{V}_t = \widetilde{P}_t \widetilde{P}_t^* \widehat{V}_t. \tag{6.4}$$

Combining the equations (6.3) and (6.4) yields

$$\begin{aligned} V_t - Q_t Q_t^* V_t &= \begin{pmatrix} P_{t_1} Z_{t_1} E_{t_1} \\ P_{t_2} Z_{t_2} E_{t_2} \end{pmatrix} + \begin{pmatrix} Q_{t_1} & \\ & Q_{t_2} \end{pmatrix} \widetilde{P}_t \widetilde{P}_t^* \widehat{V}_t \\ &= \begin{pmatrix} P_{t_1} & & Q_{t_1} & \\ & P_{t_2} & & Q_{t_2} \end{pmatrix} \begin{pmatrix} I & & \\ & I & \\ & & \widetilde{P}_t \end{pmatrix} \begin{pmatrix} Z_{t_1} E_{t_1} \\ Z_{t_2} E_{t_2} \\ \widetilde{P}_t^* \widehat{V}_t \end{pmatrix}, \end{aligned}$$

where it is important to keep in mind that $\widetilde{P}_t$ has $2k$ rows, so the dimensions of the first and second factor match.

To obtain the final result, we compute a thin Householder factorization

$$\begin{pmatrix} Z_{t_1} E_{t_1} \\ Z_{t_2} E_{t_2} \\ \widetilde{P}^* \widehat{V}_t \end{pmatrix} = \widehat{P}_t Z_t \tag{6.5}$$

with a $k \times k$ upper triangular matrix Z_t and let

$$P_t := \begin{pmatrix} P_{t_1} & & Q_{t_1} & \\ & P_{t_2} & & Q_{t_2} \end{pmatrix} \begin{pmatrix} I & & \\ & I & \\ & & \widetilde{P}_t \end{pmatrix} \widehat{P}_t.$$

Since it is a product of three isometric matrices, P_t is also isometric, so (6.1b) holds.

By our construction, we have

$$V_t - Q_t Q_t^* V_t = \begin{pmatrix} P_{t_1} & & Q_{t_1} & \\ & P_{t_2} & & Q_{t_2} \end{pmatrix} \begin{pmatrix} I & & \\ & I & \\ & & \widetilde{P}_t \end{pmatrix} \widehat{P}_t Z_t = P_t Z_t,$$

so (6.1a) holds, too.

Since (6.1c) holds for the sons t_1 and t_2, we have

$$
P_t^* Q_t = \widehat{P}_t^* \begin{pmatrix} I & & \\ & I & \\ & & \widetilde{P}_t^* \end{pmatrix} \begin{pmatrix} P_{t_1}^* & & \\ & P_{t_2}^* & \\ Q_{t_1}^* & & \\ & & Q_{t_2}^* \end{pmatrix} \begin{pmatrix} Q_{t_1} F_{t_1} \\ Q_{t_2} F_{t_2} \end{pmatrix}
$$

$$
= \widehat{P}_t^* \begin{pmatrix} I & & \\ & I & \\ & & \widetilde{P}_t^* \end{pmatrix} \begin{pmatrix} P_{t_1}^* Q_{t_1} F_{t_1} \\ P_{t_2}^* Q_{t_2} F_{t_2} \\ F_{t_1} \\ F_{t_2} \end{pmatrix} = \widehat{P}_t^* \begin{pmatrix} I & & \\ & I & \\ & & \widetilde{P}_t^* \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ \widehat{Q}_t \end{pmatrix}.
$$

We have constructed $\widetilde{P}_t$ by extending $\widehat{Q}_t$ to an orthogonal basis, so we have $\widetilde{P}_t^* \widehat{Q}_t = 0$ and conclude

$$
P_t^* Q_t = \widehat{P}_t^* \begin{pmatrix} I & & \\ & I & \\ & & \widetilde{P}_t^* \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ \widehat{Q}_t \end{pmatrix} = \widehat{P}_t^* \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} = 0,
$$

i.e., (6.1c) holds also for t. The construction is complete, and we have proven the following result:

Theorem 6.1 (Projection error) *The matrices $(Z_t)_{t \in \mathcal{T}_\mathcal{I}}$ defined by (6.2) and (6.5), respectively, satisfy*

$$
\|V_t \hat{x}_t - Q_t Q_t^* V_t \hat{x}_t\| = \|Z_t \hat{x}_t\| \qquad \text{for all } t \in \mathcal{T}_\mathcal{I}, \ \hat{x}_t \in \mathbb{R}^k.
$$

Remark 6.2 (Complexity) *The matrices P_t appearing in our construction are only required for the proof, but not for the practical algorithm. As in Remark 3.4, we assume that there is a constant C_{qr} such that a Householder vector of dimension $n \in \mathbb{N}$ can be constructed and applied in not more than $C_{qr} n$ operations.*

For a leaf cluster $t \in \mathcal{T}_\mathcal{I}$, we first apply k Householder reflections to construct $\widetilde{P}_t$. This takes not more than $C_{qr}(\#\hat{t})k^2$ operations. Applying the reflections to compute the matrix $\widetilde{P}_t^* V_t$ takes $C_{qr}(\#\hat{t})k^2$ operations. This matrix has $(\#\hat{t}) - k$ rows and k columns, so computing its Householder factorization requires $C_{qr}(\#\hat{t} - k)k^2$ operations. We obtain a total of

$$
3 C_{qr}(\#\hat{t})k^2
$$

operations for leaf clusters.

For a non-leaf cluster $t \in \mathcal{T}_\mathcal{I}$, the construction of $\widetilde{P}_t$ as a product of k Householder reflections takes $C_{qr}(2k)k^2$ operations. Setting up the matrix

$$
\begin{pmatrix} Z_{t_1} E_{t_1} \\ Z_{t_2} E_{t_2} \\ \widetilde{P}_t^* \widehat{V}_t \end{pmatrix}
$$

25

takes $2k^3$ operations for both the first and second block and $C_{qr}(2k)k^2$ for the third. The matrix has not more than $4k$ rows and k columns, so the final Householder factorization takes not more than $C_{qr}(4k)k^2$ operations. We obtain a total of

$$C_{qr}(2k + 4k)k^2 + 4k^3 = (6C_{qr} + 4)k^3$$

operations for non-leaf clusters. In the general case, i.e., if $\mathcal{T}_\mathcal{I}$ is not necessarily a binary tree, we obtain the bound

$$(3C_{qr} + 2)k^3 \#\operatorname{sons}(\mathcal{T}_\mathcal{I}, t).$$

Adding the operations for all clusters and taking Remark 2.4 into account, we conclude that preparing $(Z_t)_{t\in\mathcal{T}_\mathcal{I}}$ for all clusters takes not more than

$$3C_{qr}k^2\#\mathcal{I} + (3C_{qr} + 2)k^3\#\mathcal{T}_\mathcal{I}$$

operations. Under the assumptions of Remark 4.3, the first term vanishes, since the matrices Q_t have full rank in leaves, therefore no approximation error can occur. In this case, the computational work for preparing all matrices $(Z_t)_{t\in\mathcal{T}_\mathcal{I}\setminus\mathcal{L}_\mathcal{I}}$ is in $\mathcal{O}(k^3\#\mathcal{T}_\mathcal{I})$.

Remark 6.3 (Application to induced basis) *If we apply the conversion algorithm to obtain an efficient procedure for the matrix-vector multiplication, the cluster basis V is the induced basis, and the rank of the induced basis can be bounded by $k_A + C_{sp}k \leq C_{sp}(k_A + k)$. For large clusters, we get a complexity of $\mathcal{O}(C_{sp}^3(k_A + k)^3)$. Fortunately, for the majority of small clusters, we have $\#\hat{t} \leq k_A + C_{sp}k$, so the matrix Z_t is smaller than our worst-case estimate suggests and the entire algorithm is still reasonably efficient.*

7 Numerical experiments

We consider the application of hierarchical vectors to the task of computing eigenvectors of a matrix corresponding to a partial differential equation. In our case, we use a simple finite difference discretization of Poisson's equation on the L-shaped domain $(0, 1) \times (0, 1)\setminus[1/2, 1]\times[1/2, 1]$. The inverse is computed by standard hierarchical matrix methods [10] and then converted into an $\mathcal{H}^2$-matrix [3, Section 6.5]. The accuracy for inversion and conversion is chosen like $\mathcal{O}(1/n)$ in order to compensate for the growing condition number.

We use a variable-order polynomial basis [6] for the representation of the hierarchical vectors, where we use bicubic polynomials in the leaf clusters. The order in each coordinate direction is increase if the ratio of the extents of the bounding boxes of father and son are less than $3/5$. This approach ensures that the order used in the root cluster increases by one each time the mesh is refined.

The standard Lagrange basis is orthogonalized [3, Section 5.4] and the matrices $(\widehat{P}_t)_{t\in\mathcal{T}_\mathcal{I}}$ and $(Z_t)_{t\in\mathcal{T}_\mathcal{I}}$ for computing the coarsening and projection error are constructed. The implementation currently constructs the entire induced basis and applies the standard backward transformation given in Figure 5 instead of the optimized version given in
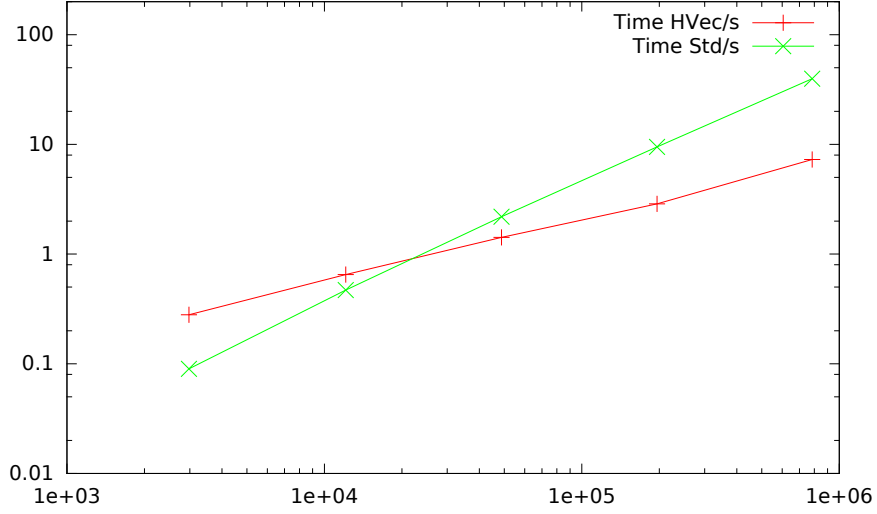
Figure 10: Runtime of the inverse iteration using standard and hierarchical vectors for different mesh resolutions

Figure 7. This choice may lead to a loss in performance for the matrix-vector multiplication algorithm, but it allows us to use standard functions to deal with the induced basis instead of implementing specialized versions for all required operations. The theoretical estimates, particularly Theorem 23, remain valid.

Once the setup is complete, we perform 20 steps of the inverse iteration using the $\mathcal{H}^2$-matrix approximation of the inverse and measure the corresponding runtime. For different refinement levels ranging from 64 to 1024 intervals per coordinate direction, corresponding to 2977 to 784897 degrees of freedom, the times are represented in Figure 10. we can see that hierarchical vectors are faster than standard vectors even for relatively small problems, and that they are faster by a factor of more than five for the largest problem.

The error tolerance for the hierarchical vector compression starts at $\epsilon = 10^{-5}$ and is approximately halved each time the mesh is refined The accuracy of the approximation is computed by converting the hierarchical vector to a standard vector and finding the Euclidean norm of the difference. Surprisingly, the results represented in Figure 11 show that the error seems to converge at a rate of $\mathcal{O}(1/n)$, while the given error tolerance decreases like $\mathcal{O}(1/\sqrt{n})$.

Next we investigate the influence of the chosen error tolerance. We fix the mesh with 784897 degrees of freedom and consider a scale of error tolerances between 5×10^{-10} and 5×10^{-7}. The leaf partitions (corresponding to $\mathcal{L}_x$) constructed by the conversion algorithm (cf. Figure 9) in combination with the adaptive coarsening (cf. Figure 4) are displayed in Figure 12. We can see that they display the typical behaviour of adaptively refined meshes: very small clusters are only used close to the singularity, while most
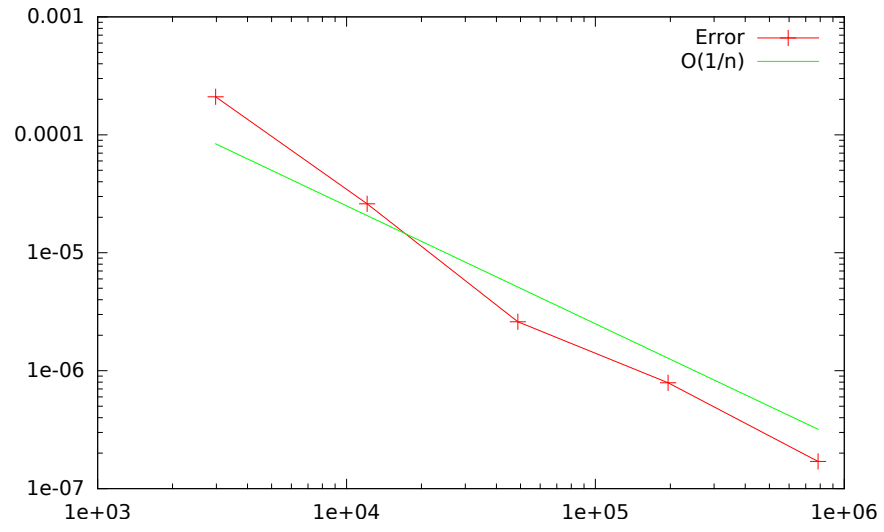
27

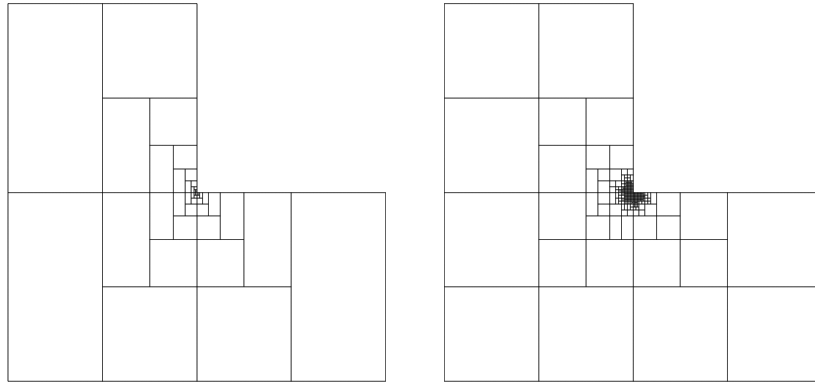Figure 11: Accuracy of the hierarchical vector for different mesh resolutions

Figure 12: Leaf partitions for error tolerances 5×10^{-7} and 5×10^{-10}
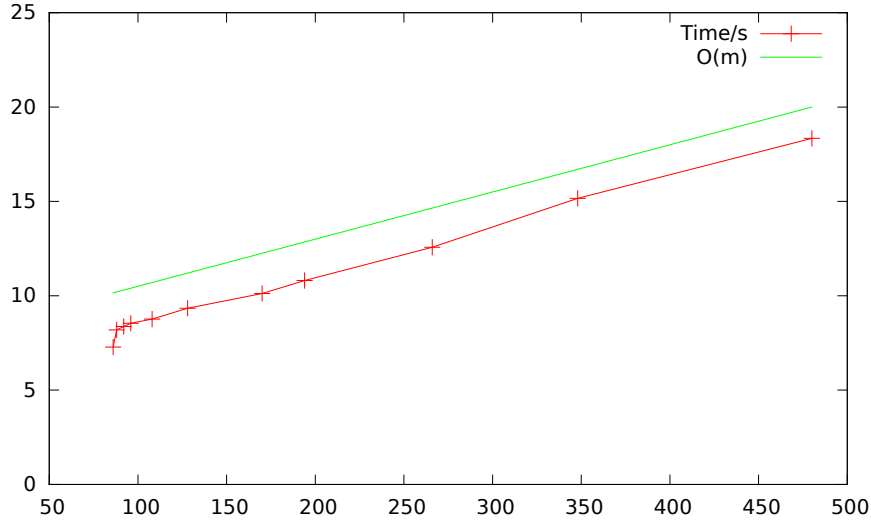
Figure 13: Computing time in relation to the number $m = \#\mathcal{T}_x$ of clusters

of the domain is covered by a few large clusters. While the algorithm requires only $\#\mathcal{T}_x = 86$ clusters to reach the tolerance 5×10^{-7}, it takes $\#\mathcal{T}_x = 480$ clusters to reach the very high tolerance 5×10^{-10}. Since we are using the standard Euclidean norm in a space of dimension 784897, the latter is already fairly close to machine precision.

Figure 13 shows the computing time for 20 steps of the inverse iteration in relation to the number of clusters in $\mathcal{T}_x$. The results confirm the prediction of Theorem 5.7: the runtime is directly proportional to $m = \#\mathcal{T}_x$.

Finally, Figure 14 shows the relation between the error tolerance and the number of clusters in the resulting subtree $\mathcal{T}_x$. The number of clusters seems to grow like $\mathcal{O}(|\log(\epsilon)|^3)$ depending on the error tolerance ϵ.

8 Conclusion and extensions

Hierarchical vectors provide us with a purely algebraic counterpart of adaptively refined meshes: clusters of indices correspond to patches of the mesh, a cluster tree corresponds to a refinement hierarchy, cluster bases play the role of local trial spaces, and the error matrices $(\widehat{P}_t)_{t \in \mathcal{T}_\mathcal{I}}$ and $(Z_t)_{t \in \mathcal{T}_\mathcal{I}}$ are used instead of local error estimators.

These error matrices provide us with the *exact* error instead of just an estimate, allowing us, e.g., to compute best approximations, either by minimizing the number of terms required for a given accuracy or by minimizing the error for a given number of terms.

The method can be applied to *any* operator that can be represented as an $\mathcal{H}^2$-matrix, e.g., to integral operators of positive or negative order and to partial differential operators or the corresponding solution operators.
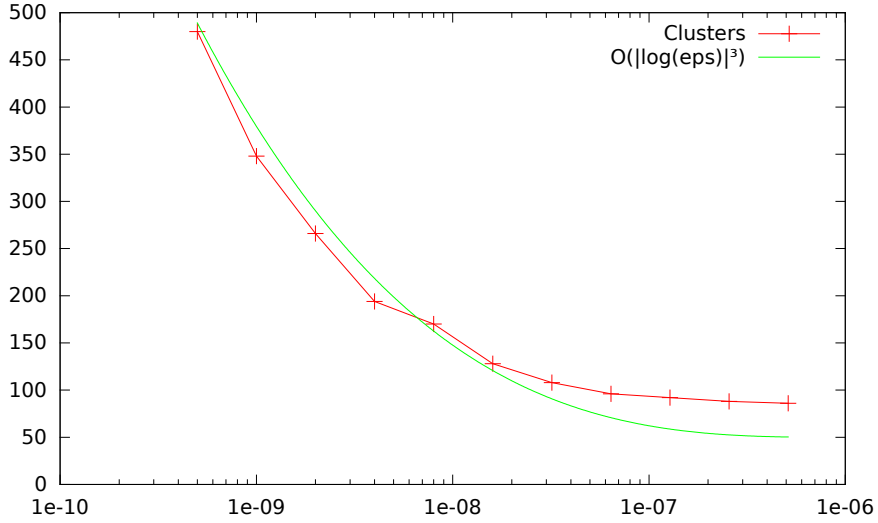
29

Figure 14: Number of clusters $\#\mathcal{T}_x$ in relation to the accuracy

The complexity of the algorithms can be bounded under the standard assumption that the block tree $\mathcal{T}_{\mathcal{I} \times \mathcal{J}}$ used for the $\mathcal{H}^2$-matrix is sparse. We have seen that in this case $\mathcal{O}(k^2 \# \mathcal{T}_x)$ operations are required to perform a matrix-vector multiplication with a hierarchical vector corresponding to a cluster tree $\mathcal{T}_x$. The numerical experiments (cf. Figure 13) confirm this estimate.

In order to obtain optimal error control, our technique relies on the projection error matrices $(Z_t)_{t \in \mathcal{T}_{\mathcal{I}}}$ used to adaptively convert the intermediate solution given in the induced cluster basis into the cluster basis $(Q_t)_{t \in \mathcal{T}_{\mathcal{I}}}$ prescribed by the application. With the direct approach presented in this paper, these matrices require $\mathcal{O}(k^2 \# \mathcal{T}_{\mathcal{I}})$ units of storage, and constructing these matrices requires $\mathcal{O}(k^3 \# \mathcal{T}_{\mathcal{I}})$ operations. Since $\# \mathcal{T}_{\mathcal{I}}$ is typically quite large compared to $\# \mathcal{T}_x$, the setup phase of the algorithm will take far more time than the actual matrix-vector multiplications.

- A long setup phase is acceptable if a very large number of matrix-vector multiplications are performed, e.g., if we are using a timestepping scheme or if we apply an iterative method to compute eigenvectors or solve optimization problems.

- If we only require an upper bound for the error, we can construct small local error matrices for each matrix block instead of the global error matrices $(Z_t)_{t \in \mathcal{T}_{\mathcal{I}}}$, or we can even just compute a bound for the norm of these matrices by a simple power iteration.

- If we are using a regular mesh, we can construct a cluster tree such that all clusters on a given level are identical up to translation. In this case, all transfer matrices and coupling matrices will also be identical on a level, so it suffices to carry out

each step of the setup only once *per level* instead of once per cluster. Under typical assumptions, this reduces the setup complexity to $\mathcal{O}(k^3 \log \#\mathcal{T}_{\mathcal{I}})$.

The experiments presented in the current paper use an $\mathcal{H}^2$-matrix approximation of the inverse. In order to improve the performance, it might make sense to replace the direct evaluation of the inverse by a iterative scheme with a preconditioner based on an $\mathcal{H}^2$-LR or $\mathcal{H}^2$-Cholesky factorization [7]. This approach would require us to compute the residual $r = b - Ax$ of the linear system, and this residual will in general not be as smooth as the solution, so the strict accuracy conditions imposed by our algorithm would lead to the subtree $\mathcal{T}_r$ corresponding to r becoming very large. Since the residual is only used to improve an approximate solution x, it might be possible to relax the accuracy conditions, e.g., by ensuring that $\mathcal{T}_r$ is constructed from $\mathcal{T}_x$ by refining each leaf cluster at most once.

References

[1] I. Babuška and W. C. Rheinboldt, *Error estimates for finite element computations*, SIAM J. Numer. Anal. **15** (1978), 736–754.

[2] E. Bänsch, *Local mesh refinement in 2 and 3 dimensions*, Impact Comp. Sci. Eng. **3** (1991), 181–191.

[3] S. Börm, *Efficient numerical methods for non-local operators: $\mathcal{H}^2$-matrix compression, algorithms and analysis*, EMS Tracts in Mathematics, vol. 14, EMS, 2010.

[4] S. Börm, L. Grasedyck, and W. Hackbusch, *Introduction to hierarchical matrices with applications*, Eng. Anal. Bound. Elem. **27** (2003), 405–422.

[5] S. Börm and W. Hackbusch, *Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices*, Computing **69** (2002), 1–35.

[6] S. Börm, M. Löhndorf, and J. M. Melenk, *Approximation of integral operators by variable-order interpolation*, Numer. Math. **99** (2005), no. 4, 605–643.

[7] S. Börm and K. Reimer, *Efficient arithmetic operations for rank-structured matrices based on hierarchical low-rank updates*, Comp. Vis. Sci. **16** (2015), no. 6, 247–258.

[8] A. Cohen, W. Dahmen, and R. DeVore, *Adaptive wavelet methods for elliptic operator equations — Convergence rates*, Math. Comp. **70** (2001), 27–75.

[9] W. Dörfler, *A convergent adaptive algorithm for poisson's equation*, SIAM J. Num. Anal. **33** (1996), no. 3, 1106–1124.

[10] L. Grasedyck and W. Hackbusch, *Construction and arithmetics of $\mathcal{H}$-matrices*, Computing **70** (2003), 295–334.

[11] W. Hackbusch and B. N. Khoromskij, *A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part II: Application to multi-dimensional problems*, Computing **64** (2000), 21–47.

[12] W. Hackbusch, B. N. Khoromskij, and S. A. Sauter, *On $\mathcal{H}^2$-matrices*, Lectures on Applied Mathematics (H. Bungartz, R. Hoppe, and C. Zenger, eds.), Springer-Verlag, Berlin, 2000, pp. 9–29.

[13] J. M. Melenk and B. Wohlmuth, *On residual-based a posteriori error estimation in hp-FEM*, Adv. Comp. Math. **15** (2001), 311–331.

[14] P. Morin, R. H. Nochetto, and K. G. Siebert, *Data oscillation and convergence of adaptive FEM*, SIAM J. Num. Anal. **38** (2000), no. 2, 466–488.

[15] C. Schwab, *p- and hp-finite element methods*, Oxford University Press, 1998.

[16] R. Stevenson, *Optimality of a standard finite element method*, Found. Comp. Math. **7** (2007), no. 2, 245–269.

[17] R. Verfürth, *A review of posteriori error estimation and adaptive mesh-refinement techniques*, Wiley, 1996.

[18] _____ , *A posteriori error estimation techniques for finite element methods*, Oxford University Press, 2013.